A path planning approach to (dis)assembly sequencing

Duc Thanh Le, Juan Cortés, Thierry Simeon

A Path Planning Approach to (Dis)Assembly Sequencing

Duc Thanh Le, Juan Cortés and Thierry Siméon

***Abstract*— x^2 The paper describes a new method for simultaneously (dis)assembly sequencing and path planning. Indeed, both are parts of a same problem that can be formulated in a general path planning framework. Based on this formulation, the algorithm proposed in this paper extends a popular sampling-based path planner, RRT, to solve general (dis)assembly planning problems involving objects with arbitrary shapes, and possibly requiring non-monotonic (dis)assembly sequences. The method does not require complex geometric computations, and is easy to implement. Experimental results show the efficiency of the method for solving a large class of problems.**

I. Introduction

Disassembly planning is a very active research field with a number of direct applications such as end-of-life product processing, maintenance operations, and product repair [1]. Moreover, integrated within CAD/CAM systems [2], disassembly planning helps to design products that are easier to manufacture, to maintain and to recycle.

Since a bijection between assembly and disassembly sequences usually exists [3], the assembly-by-disassembly strategy has been a common approach to assembly planning. Thus, although a distinction between both problems can be made [4], it is usual to talk indistinctly about assembly and disassembly planning. This paper directly addresses the latter problem. Nevertheless, like most of related works, the proposed method can be used to infer the assembly sequence from the model of the assembled object.

Disassembly planning can be tackled at different levels of detail [4]. The highest level concerns *disassembly sequencing*, which is the problem of listing subsequent disassembly actions that can separate individual parts of an assembly. This problem is usually formulated as a discrete search and optimization problem, and is solved using AI methods such as AND/OR graphs [5] or genetic algorithm [6]. Geometric reasoning approaches (e.g. [7], [3], [8]) can be applied at this level in order to reduce the combinatorial complexity of the disassembly sequencing problem. *Disassembly path planning*, which addresses the parts motion considering physical and manipulability constraints, rises at a more detailed level of the disassembly planning problem. Due to the high computational complexity of treating all part motions simultaneously, the disassembly path planning problem has usually been formulated for a single part. This simpler instance is also called the *assembly maintainability study* [9]. Efficient path planning methods (e.g. [10], [11]), based on the popular RRT algorithm [12], have been proposed to solve very constrained single-part disassembly problems on complex CAD models. Very recently, the RRT algorithm has also been extended to disassembly path planning for objects with articulated parts [13].

However, disassembly sequencing and path planning are parts of a whole problem, and ideally, they have to be treated simultaneously. The relationship between both sub-problems is more obvious for non-monotonic disassembling (see Figure 3 for an example), in which parts have to be moved to intermediate locations for permitting the disassembly of other parts. Despite their potential interest, few methods have been proposed for simultaneously disassembly sequencing and path planning in a general framework, like the one presented in this paper. Probably the most closely related method was proposed by Sundaram *et al.* [14]. Based on randomized path planning algorithms, this technique was able to compute disassembly sequences considering all the parts disassembly paths. The method samples motion directions of one or several parts using geometric information (i.e. the normal direction to faces in contact). Although general, this method strongly depends on geometric operations, and thus, its performance for solving problems involving parts with complex shapes is questionable. Besides, its ability to treat non-monotonic disassembly problems was neither experimentally proved nor discussed.

This paper introduces a general formulation for simultaneously disassembly sequencing and path planning (Section II), and proposes an algorithmic solution based upon it. The proposed method builds on sampling-based path planning algorithms, with are able to solve problems in high dimensions (i.e. involving many mobile parts). In particular, it extends the ML-RRT algorithm [13] (the principle is reminded in Section III), which was originally proposed for disassembly path planning of two objects with articulated parts. The idea developed in this paper consists in iterating the ML-RRT algorithm for extracting all the parts of a general assembly (Section IV). The performance of the planner is demonstrated on several academic examples (Section IV). Because the method does not rely on any sophisticated geometric computations but only uses collision detection, it should scale well for treating more complex CAD models, for which efficient collision checking techniques are available.

II. Problem Formulation

This section presents a unified formulation for disassembly path planning and disassembly sequencing. Disassembly planning can be formulated as a particular instance in a general path planning framework, using the notion of configuration-space [15], [16]. A configuration q is a minimal

All the authors are with the CNRS; LAAS; 7 avenue du colonel Roche, F-31077 Toulouse, France; and with the Université de Toulouse; UPS, INSA, INP, ISAE; LAAS; F-31077 Toulouse, France

set of parameters defining the location of the mobile system in the world, and the configuration-space C is the set of all the configurations. Given the initial assembled configuration q_{init}, the problem consists in finding a feasible path in C form q_{init} to a disassembled configuration q_{dis}. Note that q_{dis} may not be specified by a precise goal configuration, like in the standard path planning problem, but it can be implicitly defined by a condition based on distances between parts. Path feasibility in this context mainly involves collision avoidance. Nevertheless, constraints on the the number of hands, the possible motion directions, and optimality criteria can also be considered.

In the case of a system M involving n mobile objects m_i (i.e. the assembled parts), C is the Cartesian product of the configuration-spaces of all the objects: $C = \prod C_{m_i}$, $i = 1 \dots n$. Motions of a single object m_a, which we will call *active part*, take place in a sub-manifold $C^a_j = C_{m_a} \times q^p_j$, where q^p_j is a point in a lower-dimensional manifold $\overline{C_{m_a}} = \prod C_{m_i}$, $\forall i \neq a$. This point q^p_j represents a fixed location of all the other objects, referred to as *passive parts*. Note that for each value of q^p_j, C^a_j corresponds to a foliation leaf of C. Such a foliation of C can be made for each object $m_a \in M$ being selected as an active part.

Starting from a configuration of the whole system q_j (initially, q_j is the assembled configuration q_{init}), one part m_a can be disassembled without moving other parts if it exists a feasible path in C^a_j between $q_j = (q^a_j, q^p_j)$ and a configuration $q_{sub,a} = (q^a_{dis}, q^p_j)$, which represents a subassembly with m_a extracted from the assembly.

Considering that parts are moved and disassembled one by one (what is called a *two-handed* (dis)assembly sequence in related literature), an assembly admits a monotonic disassembly sequence if the path for disassembling each part m_a can be found in only one leaf C^a_j of its corresponding foliation, being $q_j = q_{init}$ for the first part or $q_j = q_{sub,a-1}$ for the others parts. Here, $q_{sub,a-1}$ represents a subassembly where all the previous parts in the sequence have been already extracted. Therefore, the minimal number of leaves C^a_j needed to be explored is equal to the number of parts minus one (the last part does not need to move). For non-monotonic disassembly problems, motions of parts to intermediate configurations that do not correspond with a subassembly are required. Thus, the number of leaves to be explored increases, and it is necessary to find paths in C connecting these different leaves. Disassembly sequencing within this formulation can be expressed as the problem of finding the order to select active parts m_a that minimizes the number of leaves C^a_j to be explored for finding all the disassembly sub-paths.

III. Basic ML-RRT Algorithm

The Manhattan-like RRT (ML-RRT) algorithm [13] was originally proposed for disassembly path planning of two objects with articulated parts. However, the main idea behind this algorithm is more general. It consists of partitioning the configuration parameters into an *active* subset and a *passive* subset, and treating them in a decoupled manner. Active parameters are directly handled by the planner, while passive parameters are treated only when required to expand the tree. Indeed, passive parts only move if they hinder the motion of other mobile parts (active parts or other passive ones involved in the motion).

Algorithm 1 shows the pseudo-code of the main function in ML-RRT, `Expand-ML-RRT`, which is iterated for expanding the search tree. The function `SampleConf` receives as argument the list of active parts L^a and only samples the associated parameters. Thus, this function generates a configuration q^a_{rand} in a lower-dimensional manifold of the configuration-space only involving the active parameters, C^a. Note that, if only one part is tried to be disassembled at a time, $L^a = m_a$ and $C^a = C_{m_a}$. The function `BestNeighbor` selects the node to be expanded q_{near} using a distance metric in C^a. Then, the `Expand` function performs the expansion of the selected configuration by only changing the active parameters. A greedy strategy is used here. The returned configuration q_{new} corresponds to the last valid point in the straight-line segment from q_{near} toward $\{q^a_{rand}, q^p_{rand}\}$. If the expansion is not negligible, a new node and a new edge are added to the tree. The function `Expand` also analyzes the collision pairs yielding the stop of the expansion process. If active parts in L^a collide with potentially mobile passive parts in L^p, the list of the involved passive parts L^p_{col} is returned. This information is used in the second stage of the algorithm, which generates the motion of passive parts. The function `PerturbConf` generates a configuration q^p_{rand} by randomly sampling the value of the passive parameters associated with L^p_{col} in a ball around their configuration in q_{near}. Note that, if the previous call to `Expand` has been successful, q_{near} has been updated in order

Algorithm 1: `Expand-ML-RRT`

```
input  : the model M; the search-space C;
         the current tree τ; the partition {L^a, L^p};
output : the updated tree τ;
begin
    expanded ← FALSE;
    q^a_rand ← SampleConf(C, L^a);
    q_near ← BestNeighbor(τ, q^a_rand, L^a);
    (q_new, L^p_col) ← Expand(q_near, q^a_rand);
    if not TooSimilar(q_near,q_new) then
        AddNewNode(τ, q_new);
        AddNewEdge(τ, q_near, q_new);
        q_near ← q_new;
        expanded ← TRUE;
    while L^p_col ≠ ∅ do
        q^p_rand ← PerturbConf(C, q_near, L^p_col);
        (q_new, L^p'_col) ← Expand(q_near, q^p_rand);
        if not TooSimilar(q_near,q_new) then
            AddNewNode(τ, q_new);
            AddNewEdge(τ, q_near, q_new);
            q_near ← q_new;
            expanded ← TRUE;
        L^p_col ← L^p'_col \ L^p_col;
    return expanded;
end
```

to contain the new configuration of the active parameters. An attempt is then made to generate a new node by expanding q_{near} toward $\{q^a_{rand}, q^p_{rand}\}$. Only the parts in L^p_{col} move during this tree expansion. Like for active parameters, a list $L^{p'}_{col}$ is returned by the function `Expand` when the expansion is stopped by a collision involving passive parts. If this list contains new passive parts (in relation to L^p_{col}), the progress generating passive part motions may be useful to solve problems where passive parts indirectly hinder the motion of the active ones because they block other passive parts.

The ML-RRT algorithm can take into account different difficulty/priority levels for the motion of different parts. A Gaussian distribution that translates such difficulty/priority is associated with each part. Then, the function `PerturbConf` modifies or not the configuration parameters of a passive part in L^p_{col} depending on this probability distribution. The variance of the Gaussian distribution can be adaptive, evolving with the growth of the search tree. Thus, in the first iterations, the algorithm will have low tendency to produce motions of passive parts that have been defined to have low mobility, but will incrementally move such parts if a disassembly solution path is not found with these static parts. This feature of ML-RRT is exploited in the extension presented in next section.

IV. Disassembly Algorithm

The *basic* ML-RRT is able to rapidly compute the path for extracting a part from an assembly, also producing motions of other parts if necessary. The idea developed in this paper consists in iterating the ML-RRT algorithm a number of times for extracting all the parts.

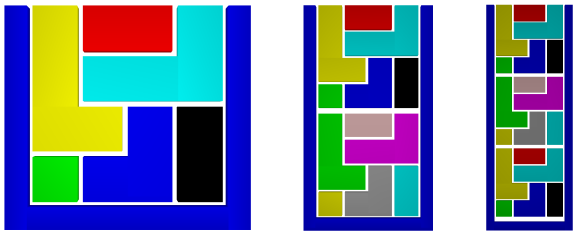
The Iterative-ML-RRT (I-ML-RRT) algorithm is sketched in Algorithm 2. At each iteration, one part is tried to be disassembled. The part is selected by the function `SelectPart`. Part selection may follow a predefined disassembly order to be tested, or maybe determined by a heuristic based on the assembly structure. If such information is not available, the part is simply selected at random. The function `SetPartition` sets the pose parameters of the selected part as active parameters, while those of all the other parts are passive. The variance of the Gaussian distribution associated with the mobility of passive parts is initialized with a high value, making these parts have low tendency to move in the first iterations. Once the parameter partition is set, the basic ML-RRT is applied to compute the disassembly path. The ML-RRT expansion process is stopped in two cases: if the currently treated part reaches a disassembled configuration (detected by the function `PartDisassembled` when the distance to all the other parts is greater than a given threshold), or if a `StopCondition` determines that the part extraction is not possible. In our implementation, the latter stop condition acts when the number of nodes in the search tree reaches a maximum user-set value *MaxTreeSize*.

The information encoded in the constructed search tree is then treated. If the current active part has been disassembled, the function `SelectEndConf` returns the last computed configuration. Otherwise, it returns the configuration that maximizes the distance to the other parts. Except if this end configuration is too similar to the initial one (i.e. parts have only slightly moved), the path connecting both configurations is computed from the search tree, and it is merged to the sub-paths obtained in previous iterations. In order to avoid including unnecessary part motions in the solution disassembly path, the threshold used in function `TooSimilar` is initialized with a high value. This value is then correlatively decreased if parts cannot be disassembled after a consecutive number of I-ML-RRT iterations. Finally, the initial configuration for the next iteration is updated to the end configuration of the current one. The whole process is iterated until all parts are disassembled, or the number of iterations reaches a maximum value, *MaxIter*. In the latter case, the algorithm returns failure.

The output of the I-ML-RRT algorithm is a path S consisting of a sequence of elementary part motions yielding the system disassembly. This path results from the concatenation of the sub-paths s_i obtained at each iteration. Each sub-path s_i involves motions of one part (the active part in the corresponding iteration), and, in some cases, slight motions of other parts that hinder its disassembly. In easy disassembly problems, most of the computed sub-paths yield the entire disassembly of one part. Nevertheless, in more difficult problems, some of the computed sub-paths may only produce partial disassembly motions that increase the clearance of the active part. The number of such intermediate sub-paths tends to increase with the difficulty of the problem. Note that, although optimality criteria are not directly considered in this work (this is a possible extension mentioned in Section VI), the I-ML-RRT algorithm tends to minimize the the number of elementary motions (i.e. the number of explored leaves C^a_j defined in Section II) required for the disassembly.

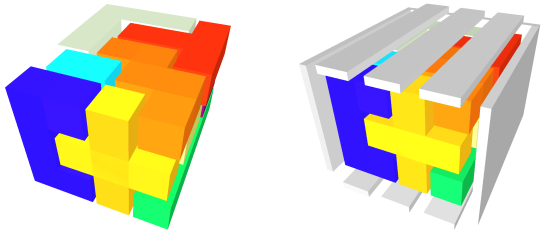
Algorithm 2: `Iterative-ML-RRT`

```
input  : the model M; the search-space C; the root q_init;
output : the disassembly pathway sequence S;
begin
    n_iter ← 0;
    repeat
        n_iter ← n_iter + 1;
        m_i ← SelectPart(M);
        (L^a, L^p) ← SetPartition(M, C, m_i);
        τ ← InitTree(q_init);
        while not StopCondition(τ, MaxTreeSize) do
            if Expand-ML-RRT(M, C, τ, L^a, L^p) then
                if PartDisassembled(L^a, q_new) then
                    break;
        q_end ← SelectEndConf(τ);
        if not TooSimilar(q_init,q_end) then
            s_i ← ExtractSubPath(τ, q_init, q_end);
            S ← MergeSubPaths(s_i);
            q_init ← q_end;
    until AllDisassembled(q_end) or n_iter ≥ MaxIter;
    if AllDisassembled(q_end) then return S;
    else return FAILURE;
end
```



(a) (b) (c)

Fig. 1. Planar Puzzles: (a) Simple, (b) Double, (c) Triple.



(a) (b)

Fig. 2. The Pentomino Puzzles: (a) without box, (b) with box.

V. Results

The I-ML-RRT algorithm has been implemented within the motion planning software Move3D [17], and tested with a set of benchmark examples. The results reported in this section aim to illustrate both the generality and the good overall performance of the method. We start with the description of the puzzle models used as examples of both monotonic and non-monotonic disassembly problems. Then, we analyze the intrinsic performance of the algorithm and show that it compares favorably with other existing techniques.

All reported results correspond to averaged values based on 50 successful runs[1] (i.e. no failure) of I-ML-RRT. The tests were performed on a AMD Opteron 2.2 GHz processor equipped with 2 Gb of memory.

A. Benchmark Models

Monotonic cases: The two first $Planar$ and $Pentomino$ models (see Figures 1 and 2) correspond to the simpler class of monotonic disassemblies in which each part can be extracted using a single continuous path (i.e. the length of the optimal disassembly sequence is equal to the number of parts, minus -the static- one). Both models are inspired from Sundaram's benchmarks [14] (Figures 1-a and 2-a) from which we created more complicate variants for the aim of our tests. First, the number of parts of the $Planar$ benchmark was increased by duplicating the original $Simple$ model into the $Double$ and $Triple$ variants (Figure 1-b,c), respectively involving 12 and 18 parts. We also considered a more difficult version or the three-dimensional $Pentomino$ puzzle in which the disassembly motions of the twelve parts are constrained by the presence of a bounding box (Figure 2-b).

Non-monotonic cases: The two other examples, $2D\_Narrow$ and the three-dimensional variant $3D\_Narrow$ (respectively shown in Figures 3 and 4), correspond to more complex problems, although less parts are involved. First, the disassembly sequence of both problems is non-monotonic, since one part has to be moved first to some intermediate position for "unlocking" the disassembly (see Figures 3-b and 4-c). Moreover, due to spatial constraints, both disassembly sequences require rotational motions for extracting the second part from the narrow corridor (see Figure 3-d) or from the assembly box (see Figure 4-e).

[1]The algorithm was run with the $MaxIter$ parameter set to a sufficiently large value (10.000), so that each run ended up when a solution was found.

B. Algorithm performance

Table I summarizes the averaged computing times obtained on the test examples. The results show the high performance of the method that is able to solve all disassembly problems in times ranging from 0.1 sec. to 13 sec. The fact that I-ML-RRT iterations may produce motions of the passive parts when extracting a given part (as opposed to a standard RRT that would act only on the active part) notably contributes to the overall performance of the algorithm, specially for the complicate case of non-monotonic problems. Note that even if the $Planar$ and $Pentomino$ puzzles can be both solved with only translational motions, the table also reports computing times obtained when rotations are allowed. These results indicate that the overhead of considering rotational motions is rather limited (factor less than two). Further results reported in Table II for the three $Planar$ puzzle variants indicate also that the computational efficiency is linearly influenced by the number of parts.

TABLE I
Algorithm Performance

Example	2D Simple Translation	2D Simple Rotation	Pentomino Translation	Pentomino Rotation	2D Narrow	3D Narrow
Nb. of parts	6	6	12	12	3	3
DOFs	12	18	36	72	9	18
Aver. CD Tests	2723	3611	18818	33026	20067	25704
Aver. Time (s)	0.1	0.16	3.79	6.55	1.65	13.83
Std. deviation	0.05	0.14	1.05	1.14	1.29	9.94

C. Effect of the disassembly order strategy

As explained in Section IV, the current implementation of the $SelectPart$ function relies on a simple random strategy for selecting at each iteration the active part tried to be disassembled. Table II further investigates the efficiency of this random choice in comparison to best and worst case selections. The best case corresponds to the correct sequence order given as input to the algorithm, while the worst case corresponds to the inverse sequence. The reported results indicate that the best/worst case selections only influence the computational efficiency by a factor less than 2.5 and that the random selection strategy stands in the middle of the time range. The table also reports the averaged number of iterations together with the number of extracted sub-paths (i.e. the length of the computed disassembly sequence). As one can see from the results, the computed solutions are close to the optimal ones for the monotonic disassemblies, but

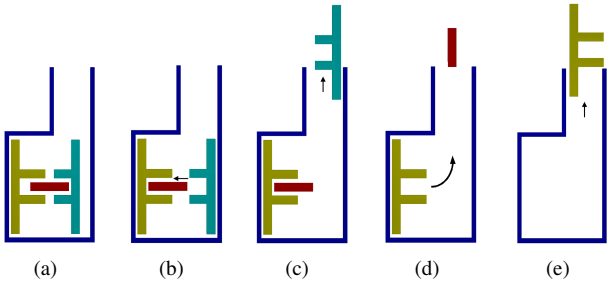


Fig. 3. 2D Narrow Puzzles: (a) Assembled configuration, (b)(c)(d)(e) Four disassembly sub-paths.

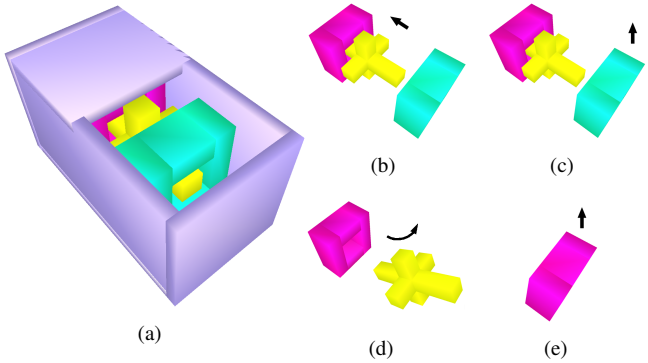


Fig. 4. 3D Narrow Puzzles: (a) Assembled configuration, (b)(c)(d)(e) Four disassembly sub-paths (the box is not displayed for clarity reasons).

remain suboptimal for the two last non-monotonic examples. While some post-processing of the solution paths could certainly filter unnecessary motions in such cases, improving the optimality of the disassembly sequence clearly remains an issue to be further investigated.

TABLE II
EFFECT OF THE SEQUENCE INITIALIZATION

Example		Nb of Elemental Motions	Average Nb of Iterations	Average Nb of Extracted Subpaths	Average Time (s)	Std Deviation
2D Simple Tran	Good-Bad	6	6-180.95	6-6.95	0.08-0.14	0.04-0.12
	Random	6	70.6	6.59	0.1	0.05
2D Double Tran	Good-Bad	12	12-212.55	12-12.56	0.66-0.88	0.06-0.37
	Random	12	180.1	12.51	0.72	0.2
2D Triple Tran	Good-Bad	18	18-318.89	18-21.61	3.45-4.52	0.38-0.34
	Random	18	217.26	18.72	3.84	0.71
3D Pentomino	Good-Bad	12	12-68.7	12-15.96	4.52-7.73	0.92-1.36
	Random	12	58.7	15.73	6.55	1.14
2D Narrow	Good-Bad	4	4-370.67	4-158.61	0.71-1.84	0.29-1.29
	Random	4	36.62	10.95	1.65	1.29
3D Narrow	Good-Bad	4	4-455.54	4-90.47	10.1-21.96	4.51-4.56
	Random	4	177.19	19.16	13.83	9.94

D. Influence of the $MaxTreeSize$ parameter

In the I-ML-RRT algorithm, the $MaxTreeSize$ parameter (see Section IV) is used to control the maximal RRT's size allowed when searching for a given part disassembly motion. When this limit is exceeded, a new part is selected and tried in turn for extraction. The maximal size of the search trees may therefore be seen as an important parameter, possibly influencing the overall efficiency of the method. The analysis on the influence of this parameter, whose results are summarized in Figure 5, indicates that it actually has a very

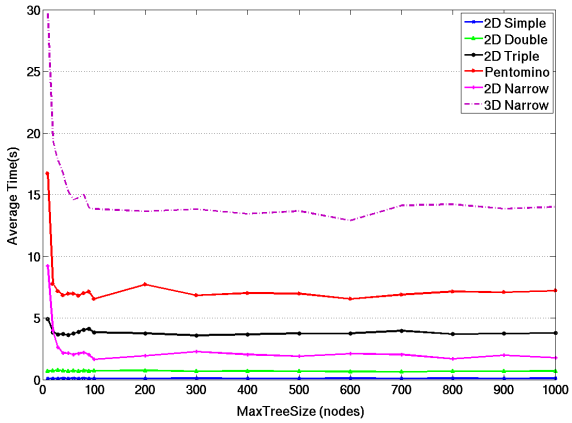


Fig. 5. Impact of $MaxTreeSize$ in the global performance of I-ML-RRT.

limited impact, and thus, it can be easily tuned. The curves of Figure 5 display for each test example the evolution of the computing time as a function of $MaxTreeSize$ for values ranging from 10 to 1000. As one can see, the performance is only affected for very small values (in the range [10,100]) resulting in search trees that are insufficiently rich for finding the whole feasible motion of a given part in a single iteration. The performance rapidly increases once the size becomes sufficient and then remains constant within most of the range (i.e. tree sizes greater than 100). Thus, the $MaxTreeSize$ parameter has to be preferably set with a large value and does not need any specific tuning since it does not really impact the overall performance. All tests reported in this section were performed with $MaxTreeSize$ set to 200.

E. Performance comparison

In this section, we compare the performance of our algorithm with two other techniques [14],[8] previously proposed for disassembly planning.

The method of Sundaram *et al.* [14] is the most closely related to ours, since it also builds on a sampling-based path planning approach. Table III relates computing times reported in [14] for the $Planar(Simple)$ and $Pentomino$ benchmarks (see Figures 1-a and 2) to the averaged times obtained with the I-ML-RRT algorithm. The reported times indicate a huge gain factor (more than 4000) in favor to I-ML-RRT. Such direct time comparison is of course biased by the higher speed of our AMD Opteron 2.2 Ghz processor compared to the older processor used in [14]. Considering a CPU speed factor of about 10 times, the algorithmic performance of I-ML-RRT remains superior by at least two orders of magnitude.

We also ran the algorithm on the difficult puzzle benchmark used in a recent publication by Fogel *et al.* [8] . This $DiagonalStar$ puzzle (see Figure 6) consists of six identical parts that create together a highly constrained assembly. Here, the difficulty of this benchmark is that it requires coordinated motions between groups of parts that need to gradually move together for disassembling the puzzle. Table IV provides some comparative results between I-ML-RRT and the exact algorithm proposed in [8] for the specific class of disassemblies solvable with infinite translational mo-

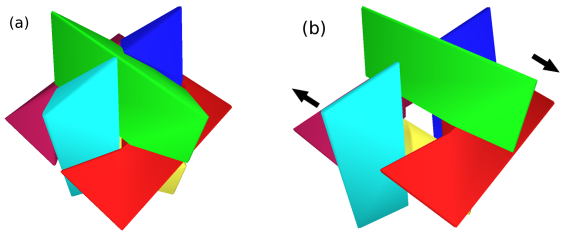


Fig. 6. The Diagonal Star Puzzle: (a) Assembled configuration, (b) Coupled disassembly motions.

TABLE III

PERFORMANCE COMPARISON WITH SUNDARAM'S APPROACH

Model	Average Time(s)	
	Iterative-MLRRT	Sundaram et al.
2D Simple	0.1	474
Pentomino	3.79	21794

TABLE IV

PERFORMANCE COMPARISON WITH FOGEL'S APPROACH

Model	Iterative-MLRRT			Fogel's algorithm		
Diagonal Star (6 parts)	Scaling factor			Nb convex sub-parts per part		
	98%	95%	90%	3	5	8
Average Time (s)	49.25	36.83	13.03	4.89	13.62	41.59

tions. As explained in Fogel's paper, the algorithm requires each part to be decomposed into convex sub-parts and the computing time strongly increases with the total number of sub-parts. On the other hand, the I-ML-RRT algorithm does not rely on exact computations of contact motions and thus requires some clearance between the parts. For this reason, we worked on slighly easier versions obtained by schrinking each part of the puzzle. Table IV compares the "shrinking factor" dependency of I-ML-RRT computing times with the "part complexity" dependency of times reported in [8]. Interestingly, although I-ML-RRT was not specifically developed for problems in which subassemblies of several parts have to slide together to disassemble each other, it is able to solve the $DiagonalStar$ puzzle almost as efficiently as the exact algorithm designed for this class of problems.

VI. DISCUSSION, CONCLUSION AND PERSPECTIVES

We have presented a general approach to disassembly planning, which treats sequencing and path planning problems simultaneously. The algorithmic solution we have developed is based on an extension of RRT, which is a simple and efficient path planning algorithm. Nevertheless, the proposed formulation can be seen as a general framework that should enable the development of other classes of algorithms.

The method has been presented considering that parts move one by one. However, active and passive parameter subsets in I-ML-RRT may involve an arbitrary number of parts. Thus, the algorithm could be directly applied for treating problems involving any number of parts simultaneously moving in different directions (i.e. disassembly problems with any number of hands). Furthermore, it will be easy to modify the method for treating more efficiently problems requiring ensemble motions of groups of parts.

The presented algorithm does not need geometric operations such as computing normals to faces for determining blocking directions. It only requires a collision checker. Consequently, I-ML-RRT can solve problems involving complex part models with any shape (convex or concave), which may be the bottleneck for other methods. However, in particular cases (e.g. polyhedral objects), there is place for improvement by integrating some geometric operations that will provide good heuristics for selecting parts to move and suitable motion directions. Indeed, a possible trend for future work would be to combine I-ML-RRT with automated geometric reasoning methods for assembly partitioning and disassembly sequencing (e.g. [7], [3], [8]).

Another future extension would be to consider optimality criteria, in particular, for minimizing the number of elementary motions. Although this criterion is already indirectly considered by the algorithmic design of I-ML-RRT (which does not keep part motions below a given distance threshold), it reminds an important issue that merits a more direct treatment. Besides, some costs associated with the disassembly could be considered within the path planning process. One possible way could be to integrate ideas of T-RRT [18].

REFERENCES

[1] A. Lambert and S. Gupta, *Disassembly modeling for Assembly, Maintenance, Reuse and Recycling*. Florida: CRC Press, 2005.

[2] E. Arai and K. Iwata, "CAD system with product assembly/disassembly planning function," *Robotics and Computer-Integrated Manufacturing*, vol. 10, pp. 41–48, 1993.

[3] D. Halperin, J. Latombe, and R. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, pp. 577–601, 2000.

[4] A. Lambert, "Disassembly sequencing: A survey," *Int. Journal of Production Research*, vol. 41, pp. 3721–3759, 2003.

[5] ——, "Optimum disassembly sequence generation," *Proc. SPIE Conf. on Environmentally Consious Manufactoring*, pp. 56–67, 2000.

[6] E. Kongar and S. Gupta, "Genetic algorithm for disassembly process planning," *Proc. SPIE Int. Conf. on Environmentally Conscious Manufactoring II*, pp. 54–62, 2001.

[7] R. Wilson, L. Kavraki, T. Lozano-Perez, and J. Latombe, "Two-handed assembly sequencing," *Int. Journal of Robot. Research*, vol. 14-4, pp. 335–350, 1995.

[8] E. Fogel and D. Halperin, "Polyhedral assembly partitioning with infinite translations or the importance of being exact," *Proc.Workshop on the Algorithmic Foundations of Robotics*, 2008, in press.

[9] H. Chang and T.-Y. Li, "Assembly maintainability study with motion planning," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 1012–1019, 1995.

[10] E. Ferré and J. Laumond, "An iterative diffusion algorithm for part disassembly," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 3194–3150, 2004.

[11] I. Aguinaga, D. Borro, and L. Matey, "Parallel RRT-based path planning for selective disassembly planning," *Int. J. Adv. Manuf. Techno.*, vol. 36, pp. 1221–1233, 2008.

[12] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees : Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. Donald, K. Lynch, and D. Rus, Eds. Boston: A.K. Peters, 2001, pp. 293–308.

[13] J. Cortés, L. Jaillet, and T. Siméon, "Disassembly path planning for complex articulated objects," *IEEE Trans. Robot.*, vol. 24, pp. 475–481, 2008.

[14] S. Sundaram, I. Remmler, and N. Amato, "Disassembly sequencing using a motion planning approach," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 1475–1480, 2001.

[15] J. Latombe, *Robot Motion Planning*. Boston: Kluwer Academic Publishers, 1991.

[16] S. LaValle, *Planning Algorithms*. New York: Cambridge University Press, 2006.

[17] T. Siméon, J.-P. Laumond, and F. Lamiraux, "Move3D: A generic platform for path planning," *Proc. IEEE Int. Symp. Assembly and Task Planning*, pp. 25–30, 2001.

[18] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based RRT for path planning in continuous cost spaces," *Proc. IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, pp. 22–26, 2008.