Continuous Collision Detection of Pairs of Robot Motions Under Velocity Uncertainty

Edvin Åblad, Domenico Spensieri, Robert Bohlin, and Ann-Brith Strömberg

***Abstract*—In automotive manufacturing, production systems typically involve multiple robots and, today, are being individualized by utilizing the concept of digital twins. Therefore, the robot programs need to be verified for each individual product. A crucial aspect is to avoid collisions between robots by velocity tuning: This involves an efficient analysis of pairs of robot paths and determining if swept volumes of (sub) paths are disjoint. In general, velocity uncertain motions require disjoint sweep volumes to be safe. We optimize a clearance lower bounding function to provide new sample points for clearance computations. Due to the computational cost of each distance query, our sampling strategy aims to maximize the information gained at each query. The algorithm terminates when robot paths are verified to be disjoint or a collision is detected. Our approach for disjoint paths is inspired by the technique for continuous collision detection known as *conservative advancement*. Our tests indicate that the proposed sampling method is reliable and computationally much faster than creating and intersecting octrees representing the swept volumes.**

***Index Terms*—Conservative advancement (CA), continuous collision detection (CCD), Lipschitz optimization, Manhattan distance, multiple robots, Smart Assembly 4.0, swept volumes.**

I. Introduction

MOTION planning and coordination algorithms spend much time checking whether a robot's path is acceptable (collision-free) w.r.t. the environment and/or another robot's path. Hence, key ingredients in these algorithms are efficient ways of determining whether paths, or pairs of paths, are collision-free.

Ensuring that a robot path is collision-free w.r.t. a static environment is, however, a current challenge. As an example, checking that certain waypoints are free is not sufficient; special care is required to ensure that the entire path is collision-free, regardless of the planning method used, being it based on probabilistic roadmap [1], [2], rapidly exploring random tree [3], feedback-based information roadmap [4], or mixed integer linear programming (MILP) [5].

Manuscript received July 4, 2020; accepted December 26, 2020. Date of publication January 25, 2021; date of current version October 1, 2021. This work was supported in part by the project Smart Assembly 4.0, in part by the Swedish Foundation for Strategic Research (SSF), and in part by the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology. This article was recommended for publication by Associate Editor J. O'Kane and Editor M. Yim upon evaluation of the reviewers' comments. *(Corresponding author: Edvin Åblad.)*

The authors are with the Geometry and Motion Planning Group, FraunHofer-Chalmers Research Centre for Industrial Mathematics, 41288 Gothenburg, Sweden (e-mail: edvin.ablad@fcc.chalmers.se; domenico.spensieri@fcc.chalmers.se; robert.bohlin@fcc.chalmers.se; anstr@chalmers.se).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2021.3050011.

Digital Object Identifier 10.1109/TRO.2021.3050011

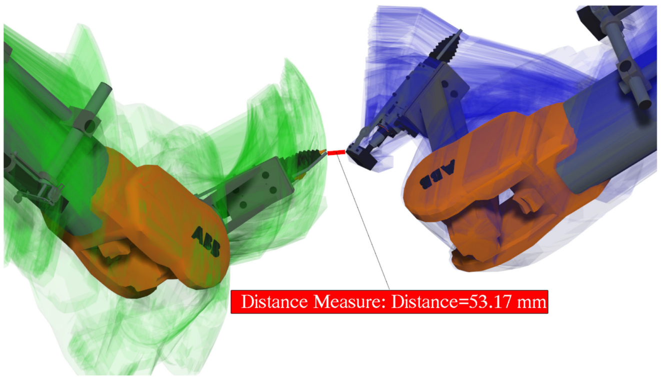

Fig. 1. Illustration of the minimal clearance between the sweeps of two industrial robots traveling along their weld task interpaths. The paths are said to be disjoint if the minimal clearance exceeds a specified threshold.

The same issue applies when checking if two independent robot motions are collision-free with respect to each other, i.e., their swept volumes are disjoint; see Fig. 1. Usually, each robot has its own control system; therefore, the relative positioning of two robots along their motions cannot be guaranteed continuously. This means that their geometric paths are known, but their trajectories are uncertain. We call this velocity uncertainty in the trajectories, in contrast to the positioning one. If positioning accuracy is also low, then it is not enough to check that their paths are collision-free, but they are safe if being a distance always greater than a certain threshold value, accounting for modeling positioning uncertainty. This is what we model in this article and propose an efficient computational algorithm to verify.

The seminal paper of O' Donnell and Lozano-Pérez [6] provided a method to coordinate robot motions in time based on the assumption of trajectory unpredictability. Their target applications were arc welding where the speed might be adjusted in response to observed weld parameters or, more in general, when some robot operations involve a sensor-based duration change. One of the main reasons to guarantee two robot paths between the two respective waypoints to be disjoint is unpredictable stops of the robot in the middle of a motion. Note that this is typical since enforcing the entire paths to be disjoint is very conservative; it suffices that certain short subpaths are disjoint to construct safe trajectories.

Another reason to consider velocity uncertain trajectories is to simplify the multirobot path planning problem by planning each robot path independently. And in order to prevent collision among robots, a fixed-path coordination [7, ch. 7.2.2] is employed; cf., [8]–[10]. In [8], subpaths are checked for being disjoint in order to decide optimal trajectory velocities.

Another aspect of velocity uncertainty is the robustness of a solution. If the robots can deviate for a nominal trajectory without causing collision, we can consider safe modifications of the trajectories to, e.g., minimize the energy usage of the robots [11], or we can consider uncertain trajectories, e.g., a robot being human agent [12]. Finally, the geometric operation of checking that two (sub) paths are disjoint can be an important tool when analyzing a solution.

Moreover, factories—automotive in particular—are currently changing from a mass production to a product individualized workflow. The challenges of this transformation are considered in the project Smart Assembly 4.0 (SA4.0) that utilizes the concept of digital twins [13]. Here, every product item has its own digital representation, including geometrical variations. Hence, nominal paths of the industrial robot manipulators need to be verified online, and possibly modified and coordinated, to comply with the specific product item geometry. Hence, an efficient routine to verify subpaths to be disjoint is our main goal of this research.

A. Article Outline

We propose a novel sampling method for determining whether a pair of robot paths is disjoint, which also specializes to a collision check for motions. This method is inspired by the well-established technique of *conservative advancement* (CA), cf., [14], and inherits the crucial property that it is exact, i.e., without false-positive or true-negative outcomes. Its main enabler is the efficient optimization of a clearance lower bounding function.

Section II presents published methods related to either the collision-free or the disjoint query and current works utilizing those methods. Our contributions are derived in Section III: the clearance bounding function, the disjoint query, analysis of computational complexity, generalizations to identify disjoint subregions, and first/last points of collision. For reference, Section IV shortly presents a well-established method for the disjoint query based on an approximation of the robot paths' sweeps. Section V presents test instances, numerical comparisons of the disjoint query, and its two generalizations. Section VI concludes this article.

B. Scope and Limitations

Our algorithm utilizes that the robot paths are parametrized with a maximum velocity of 1 m/s, called a *unit-velocity* parametrization; see Definition 1. However, as the derivation of such a parametrization depends on the robot type, it is outside the scope of this contribution. The parametrization is usually the product of a path sensitivity analysis to estimate a so-called Lipschitz constant(s) that bounds the maximum velocity of any part of the robot. This can be done either by a sampling or analytical method, typically assuming a specific robot type. For example, [15] presents bounds on the displacement of robots consisting of prismatic and revolute joints; a similar idea is explained in [7, ch. 5.3.4].

Hence, our algorithm is applicable to any type of robot path for which the unit-velocity parametrization can be retrieved. The robots can be, e.g., nonconvex rigid bodies, a union of rigid bodies, or even nonrigid. We consider methods based on more specific assumptions on the robots and their paths to be outside the scope of this contribution. Moreover, by the motivation of SA4.0, we focus on an assembly cell in the automotive factory. Hence, our computational experiments and results are based on such instances.

Definition 1. (Unit-velocity parametrization): A parametrization of a robot path that bounds the maximum velocity of any point on the robot by 1 m/s.

II. Related Work

There are several algorithms for checking collision between moving objects, i.e., continuous collision detection (CCD). Typically, these algorithms address the problem with sampling, i.e., ensuring that a continuous motion, and not only certain points of it, is free from collision. Three main approaches are: CA, different sweep representations, and to build a custom bounding volume hierarchy (BVH) based on the trajectory. Such algorithms are then used by motion planning algorithms and libraries (e.g., [16]) to validate that the used paths are collision-free.

Some path planning algorithms use collision-free paths provided by a low-level motion planner and consider the multirobot path planning problem defined in a discrete graph. For example, in [17], the robots collide if they simultaneously either occupy the same vertex or traverse the same edge in opposite directions. However, they do not consider the case when two different vertices correspond to robot poses that collide. In [18] and [19], we propose an algorithm for resolving this issue by enforcing a space partition of the workspace. The article [21] suggests a method for solving the multirobot planning problem on a graph with complicated constraints: The framework enables the formulation of a static space partition such that certain pairs of vertices or edges are not allowed to be traversed by different robots. However, the framework is not suitable for time-dependent constraints.

Anyway, the fundamental problem of CCD still needs to be addressed at some level of any path planning algorithm. One of the most popular approaches for a single path is that of using a BVH, which is dynamically refined: When a collision is detected, the BVH is rebuilt on smaller time intervals and the collision detection continues recursively; see [7]. The authors of [22] present a CCD based on BVH using line swept spheres (LSS). The method applies multiple levels of approximations, using higher precision for links of the robots (i.e., rigid bodies) only when needed; the main test uses interval arithmetic and motion bounding functions to check whether oriented bounding boxes (OBBs) are in collision during the motion. The time interval is subdivided in order to increase the fitness of the bounds; at termination, if the OBBs still overlap, an exact query is executed for the primitives contained in the OBBs; see [23].

Another popular approach for CCD is that of CA, which basically utilizes the Lipschitz constant of a robot path to determine a safe traversal along the path. An early example of CA was suggested in [14], which uses an adaptive sampling technique. In [24], a similar technique is used, but it assumes a bounded acceleration to derive a safe step length. In [25], it is illustrated how CA can be improved by not sampling the path from start to end but using, e.g., a binary strategy (cf., [26]) that checks equally many points if the segment is collision-free but has a chance of finding an existing collision with fewer samples. This idea is adopted in [27], which claim that for their robot, this approach is superior to generating or approximating the swept volume. The chapter [15] presents an example of CA and adaptive sampling, which uses a binary strategy rather than considering the trajectory sequentially; it also utilizes that the distance at the static position need not be computed exactly but only bounded from below. Another idea is to combine CA and BVH, using the BVH to exclude paths that are clearly disjoint, and using CA to find the first point of contact; cf., [28].

The idea of CA and binary search has also been applied within sampling-based motion planning algorithms. In [29], the (lower bounds of) clearances at two discrete configurations are used to deduce if the connecting geodesic segment is collision-free. Moreover, utilizing a maximum robot displacement caused by a configuration perturbation has also been incorporated in path planning algorithms, e.g., in [30], and in [31], the direction of movement is also accounted for.

A straightforward approach is to first build an approximation of each robot's paths sweep at a desired accuracy and then determine if this sweep intersects the static geometry or another sweep. This is done in [10] to detect if two quadcopter paths can be safely traversed regardless of time. The authors consider the intersection of sets of points swept by the robots. The paper [32] proposes an adaptive approximation of the swept volume based on axis-aligned boxes (AAB) containing the swept volume; the AABs are derived from the trajectory using interval arithmetic. The paper [33] studies a robot for which simple primitives approximate its volume well; the sweep of these primitives is analytically derived and checked for collisions against a world octree, containing all static objects. In [34], octrees for each moving object are generated and then checked for collisions against a world octree.

The type of collision detection algorithm to use is usually determined by the application. Some applications accept crude models of their objects, e.g., ellipsoid of quadcopters [10]. Others require much more precise models, e.g., robotic arm and tool in an assembly process [8]. In our case, when the robots are forced to work close to each other and to the environment, more accurate collision models are appropriate while crude conservative approaches are inapplicable.

Our work assumes that a unit-velocity parametrization (Definition 1) can be retrieved, similar to that in [4], which uses it for complexity analysis rather than to guarantee collision-free motions. In [4], it is assumed that the sampled points are dense enough. However, the motions include an uncertainty, and path collision probabilities are estimated by a path sampling method; hence, an exact CCD might be computationally too heavy and its outcome is not so important.

One of our main contributions is the efficient minimization of the clearance lower bounding function, as derived directly from the clearances at discrete robot poses and from the Lipschitz constant that determines the unit-velocity parametrization. In [35] and [36], the optimization of an unknown function, where only a Lipschitz constant is assumed to be known, is studied. These works employ, however, the Euclidean norm, which is a common choice; cf., [37]. In our application, in which two separate motions contribute to the uncertainty, the L^1 norm is instead retrieved. It should be noted that our lower bounding function based on the L_1 norm is closely related to a Voronoi diagram using the Manhattan distance and additively weighted input sites. This is due to the global minimum being retrieved at some edge of this Voronoi diagram. Such a diagram is rather nonstandard; e.g., [38] considers it only briefly and gives no hints on incremental algorithms. However, the Manhattan distance Voronoi diagram can be incrementally created in $\mathcal{O}(n \log n)$ time; see, e.g., [39].

To the best of our knowledge, there are few examples of efficient exact methods for verifying a pair of robot paths disjoint, apart from constructing their respective sweeps, which is often computationally expensive. We investigate how a method based on CA compares to a method based on sweeps. Our method also provides efficient detection of the first and last points of collision, i.e., when one robot enters the other's sweep. Moreover, it utilizes and improves a lower bounding function to determine where, along the paths, to sample. This preserves all information from previous clearance computations. Note that our algorithm can be specialized to instead detect collisions between pairs of robot motions; it then resembles the method of using a binary search from [15].

III. Path–Path Collision Test

We say that two paths are disjoint if the robots' swept volumes do not intersect or, more generally, are separated by a specified threshold. Specifically, let the clearance $g(t_A, t_B)$ be the Euclidean distance between two robots, A and B, positioned at time t_A and t_B along their corresponding parametrized paths. Then, the paths of two robots, A and B, are disjoint if and only if

$$g(t_A, t_B) > \tau, \quad (t_A, t_B) \in D \tag{1}$$

where $\tau \in \mathbb{R}_+$ is a given threshold and $D := [0, T_A] \times [0, T_B]$ is the time domain. Note that verifying (1) corresponds to exploring the rectangular domain D, i.e., find a point $(t_A, t_B) \in D$ violating (1) or verify that no such point exists.

However, D contains infinitely many points; thus, guaranteeing that all points in D are collision-free requires additional assumptions on the paths and their parametrizations. We assume that each robot path admits a unit-velocity parametrization (recall Definition 1). Consequently, without loss of generality, we can assume that the domain D is composed by two such rescaled

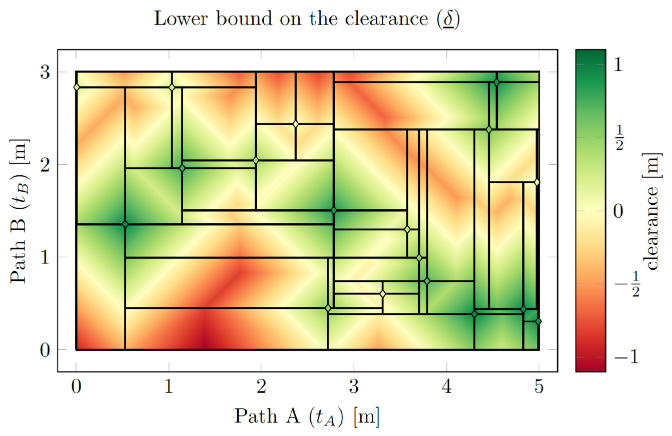

Fig. 2. Domain representing the two paths is decomposed into regions, within each of which the piecewise linear lower bounding function is convex. The black lines denote region boundaries and the rhombus-shaped markers represent collision-free points in which the function g is evaluated.

Algorithm 1: Verify that a Pair of Robot Paths are Disjoint.

```
1:  procedure $L^1$-min sampling$D, \tau$
2:    input: A domain $D$ and a threshold $\tau$.
3:    output: Whether (1) holds.
4:    $N := 1$
5:    $\bar{\boldsymbol{t}}_1 := (T_A, T_B)/2$
6:    repeat
7:      $\bar{\delta}_N := g(\bar{\boldsymbol{t}}_N)$
8:      if $\bar{\delta}_N \leq \tau + \varepsilon$ then
9:        return false
10:     end if
11:     $\underline{\delta}(\boldsymbol{t}) := \max\{\underline{\delta}(\boldsymbol{t}),\ \bar{\delta}_N - \|\boldsymbol{t} - \bar{\boldsymbol{t}}_N\|_1\}$
12:     $N := N + 1$
13:     $\bar{\boldsymbol{t}}_N := \arg\min_{\boldsymbol{t} \in D} \underline{\delta}(\boldsymbol{t})$
14:   until $\underline{\delta}(\bar{\boldsymbol{t}}_N) > \tau$
15:   return true
16: end procedure
```

paths. Thus, the clearance function g is nonsmooth, nonconvex, and it admits a unit L^1 Lipschitz constant.

Using that the maximum velocity of the robots equals 1, we can determine regions in which the paths are disjoint. For instance, given N collision-free points $\bar{\boldsymbol{t}}_i := (\bar{t}_A, \bar{t}_B)_i$ with clearances $\bar{\delta}_i := g(\bar{\boldsymbol{t}}_i) > \tau$, $i = 1, \ldots, N$, we retrieve a lower bounding function on the clearance, $g(\boldsymbol{t})$, namely

$$\underbrace{\max_{i=1,\ldots,N} \{\bar{\delta}_i - \|\boldsymbol{t} - \bar{\boldsymbol{t}}_i\|_1\}}_{=:\underline{\delta}(\boldsymbol{t})} \leq g(\boldsymbol{t}), \quad \boldsymbol{t} \in D. \tag{2}$$

Hence, the (rhombus-shaped) regions defined by

$$\{\boldsymbol{t} \in D \mid \bar{\delta}_i - \|\boldsymbol{t} - \bar{\boldsymbol{t}}_i\|_1 > \tau\}, \quad i = 1, \ldots, N$$

are guaranteed to be free of collision. Note that these regions are square but rotated; thus, we refer to them as rhombuses.

We can thus argue that by evaluating the clearance function g in finitely many points, we can either find a point corresponding to a collision or conclude that the paths are disjoint. This follows by the practical assumption that the distance test (1) uses a tolerance $\varepsilon > 0$; hence, $g(\bar{\boldsymbol{t}}_i) \leq \tau + \varepsilon$ means a collision. Thus, each point that is collision-free implies a free $2\varepsilon^2$ m^2 rhombus shaped area; it can be verified that less than $\lceil \varepsilon^{-1} \rceil^2$ points are sufficient to verify the disjointness criterion (1).

A. Main Algorithm

Since each evaluated point involves both a kinematic evaluation of the robots and a distance query between the robots, the number of samples should be rationed. A sampling method can either choose samples close to near collision points or assume that the paths are disjoint and choose samples far from previous points in order to gain as much information as possible (minimizing the overlap with the known collision-free regions). We suggest a tradeoff utilizing the information from the lower bounding function (2), which is illustrated in Fig. 2. Namely, to choose the global minimum of the clearance lower bound since it is a promising candidate for a collision point, and it is also far from all previous samples. Moreover, we get two easily checked termination criteria: 1) a collision-point is found and 2) the lower bound is above τ in the minimum point, and, thus, (1) holds and the paths are disjoint.

Note that Algorithm 1 contains the following two nontrivial statements:

1) the evaluation of g that includes the kinematic evaluation as well as the clearance computation;
2) the minimization of $\underline{\delta}$ defined in (2).

Since the first is a well-studied problem (see [7], [40], and [41]), we consider here the minimization problem; note, however, that—as in [15]—the algorithm allows g to be bounded from below if $g(\bar{\boldsymbol{t}}_N) > \tau$; see Section III-G.

The function $\underline{\delta}$, illustrated in Fig. 2, is clearly nonconvex and thus nontrivial to minimize. However, D can be partitioned into rectangular regions D_k in each of which $\underline{\delta}$ is convex and piecewise linear. This can be done iteratively, starting with $D_1 := D$; then for each new point $\bar{\boldsymbol{t}}_i$ included in (2) (on line 11 of Algorithm 1), find k such that $D_k \ni \bar{\boldsymbol{t}}_i$ and search along the vertical and horizontal lines through $\bar{\boldsymbol{t}}_i$ until the index i no longer defines the maximum term of (2). During this search, divide all visited regions (including D_k) by these lines, and let the new subregions replace the divided ones; cf., Fig. 2.

Given the resulting partition of D, a global minimum of the function $\underline{\delta}$ can be obtained by minimizing $\underline{\delta}$ in each region D_k. In which, we have that (cf., Fig. 2)

$$\underline{\delta}(\boldsymbol{t}) = \max_{i,j \in \{0,1\}} \{\underline{\delta}_{ij}^k - \|\boldsymbol{t} - \boldsymbol{t}_{ij}^k\|_1\}, \quad \boldsymbol{t} \in D_k \tag{3}$$

where $\underline{\delta}_{ij}^k := \underline{\delta}(\boldsymbol{t}_{ij}^k)$ and $\boldsymbol{t}_{ij}^k$ is a corner of D_k. Note that (3) is given by four supporting hyperplanes and the minimum must be supported by a diagonal pair. Due to linear dependency, the intersection line of such a pair has a constant level and $\|\boldsymbol{t} - \boldsymbol{t}_{0j}\|_1 + \|\boldsymbol{t} - \boldsymbol{t}_{1(1-j)}\|_1 = \Delta T_A^k + \Delta T_B^k$, $j = 0, 1$, where ΔT_A^k and ΔT_B^k denote the width and height of the region D_k, respectively. Equating the levels of these lines yields

$$\min_{\boldsymbol{t} \in D_k} \underline{\delta}(\boldsymbol{t}) = \max_{j \in \{0,1\}} \frac{\underline{\delta}_{0j}^k + \underline{\delta}_{1(1-j)}^k - \Delta T_A^k - \Delta T_B^k}{2}. \tag{4}$$

Thus, to access the global minimum, a heap-based priority queue, sorted by the regions' minimum values, can be used and updated.

The computational complexity of the L^1-min sampling strategy can now be analyzed. Each iteration includes one evaluation of g, an update of the lower bounding function $\underline{\delta}$, and a computation of the next minimum position $\bar{\boldsymbol{t}}_N$. While evaluating g is expensive, it is, however, independent of the number N of evaluated points and the number M of regions. An update of the lower bounding function $\underline{\delta}$ requires finding the M_N regions that are affected by the new measurement $\bar{\delta}_N$ by using an adjacency graph of the vertices and a binary tree of the regions: This is an $\mathcal{O}(\log M + M_N)$ operation. Updating the keys in the priority queue for these regions requires $\mathcal{O}(M_N \log M)$ operations and accessing the minimum region requires only $\mathcal{O}(1)$ operations. Moreover, the number M_N of affected regions D_k cannot be large since only a neighborhood of $\bar{\boldsymbol{t}}_N$ that cannot include any previous $\bar{\boldsymbol{t}}_i$'s needs to be updated. Since this neighborhood is independent of all other regions, it follows that M_N is bounded by a constant; by a similar argument, it holds that $\mathcal{O}(M) = \mathcal{O}(N)$. Thus, each iteration involves the evaluation of g and $\mathcal{O}(\log N)$ operations to update the function $\underline{\delta}$.

B. Numerical Results on Clearance Bound Minimization

The below numerical investigation of Algorithm 1 is aimed to 1) establish the argued time complexity and 2) compare it with a standard MILP optimization procedure. To this end, we will present some CPU times; see Section V for system specifications.

To test the time complexity of our algorithm for minimizing $\underline{\delta}$, we use the arguable worst possible clearance function $g(\boldsymbol{t}) := \tau + c$ since the entire region will need to be carefully explored for small values of $c > 0$. Note that any sampling strategy requires at least $N \geq \frac{T_A T_B}{2c^2}$ iterations, where equality holds when none of the collision-free rhombuses overlap. Choosing the domain $D := [0, 60] \times [0, 30]$ and $c := 0.05$ yields $N \geq 3.6 \cdot 10^5$, which is a large enough value to visualize the computational complexity.

The test reveals that the routine is fast, requiring only a few seconds to complete $8 \cdot 10^5$ iterations, which is several magnitudes faster than distance queries involving complex geometry. Moreover, the argued time complexity of $\mathcal{O}(N \log N)$ fits slightly better than, e.g., $\mathcal{O}(N \log^2 \tilde{} N)$. Fitting the measured values to $\mathcal{O}(N \log^k N)$ by minimizing the sum of squared errors yielded $k = 1.12$. That the fit is not perfect may be due to a number of practical reasons, such as N not being large enough, cache misses (i.e., memory access is not an $\mathcal{O}(1)$ operation), or memory reallocations (a sequential data storage was used to enhance cache locality).

From the test, we also noted that $N \approx 8 \cdot 10^5$ iterations are required for this case; it is roughly twice the potential minimum. This might seem as target for improvement, but it is quite good due to the following two reasons.

1) Since an evaluation is required to get information of g, some overlap is inevitable, e.g., consider a divide-and-conquer algorithm. Cover a square domain (e.g., $[0, T_B] \times [0, T_B]$) with a rhombus, then, recursively divide the rhombus into four equally sized rhombuses, until its area is sufficiently small (less than $2c^2$ in the above example). To cover the entire domain D, $N \geq 4^{\lceil \log_4 \frac{T_B T_B}{c^2} \rceil} \approx 10^6$, such rhombuses are needed—for another size of square, there is an expected factor $\mathrm{E}[4^\theta]$ of too many samples, a uniform $\theta \in [0, 1]$ yields a factor of $\frac{3}{\ln 4} \approx 2.16$—i.e., there is a lot of overlap in the last subdivision. This illustrates the issue that the optimal rhombus region area ($2c^2$) is only implicitly available in the algorithm.
2) In our sampling method, overlaps start occurring already at iteration $2 \cdot 10^5$, which is good, since, if g is not a dummy function but involves a clearance computation, then tight bounds imply that g can be computed approximately; see Section III-G. Thus, the L^1-min sampling strategy can be interpreted as $2 \cdot 10^5$ expensive iterations (search for a collision point) and $6 \cdot 10^5$ cheaper iterations (verify disjointness).

For comparison, consider computing the global minimum of $\underline{\delta}$ using the Gurobi MILP solver [42] on line 13 in Algorithm 1. Several MILP models were tested, and none competed with our suggested algorithm. The most promising model had $2N + 2$ binary variables to select an x and y interval between input points; within these intervals, the problem is linear since the L_1-norms in (2) become linear. However, already 1000 clearance points resulted in several hours of computation time. Another tested formulation was perfect (i.e., integer restrictions are redundant); it decomposed over the $(N + 1)^2$ regions composed by the above x and y intervals. This model reduces to solving $(N + 1)^2$ LP problems, each containing three variables and N constraints. However, these $(N + 1)^2$ LP problems have analytical solutions, namely (4); hence, this decomposition approach has a potential time complexity of $\mathcal{O}(N^2)$.

As noted in Section II, it is possible to minimize $\underline{\delta}$ by maintaining the Manhattan distance Voronoi diagram with additively weighted input sites (i.e., $\bar{\boldsymbol{t}}_i$ weighted by $\underline{\delta}_i$). However, to the best of our knowledge, no such algorithm is directly available, and the special case of uniform weights leads to a complexity of $\mathcal{O}(N \log N)$; cf., [39]. Hence, this idea is useful only if it is the minimization of $\underline{\delta}$ that is the bottleneck, e.g., if g is very fast to evaluate.

C. Initializing the Disjoint Query

In our L^1-min sampling strategy for verifying the condition (1), the initial sample is chosen to be in the center of the domain D. This seems like a reasonable choice since if we sample close to the boundary, some information (collision-free rhombus region) might be outside D and thus useless. However, an immediate consequence of using the center as starting point will be that the next four samples will be located in the corners of D, hence resulting in the very situation that should be avoided.

We suggest two remedies to this issue: one general approach and one approach for the case of cyclic boundaries.

The general approach relies on the fact that the lower bounding function $\underline{\delta}$ defined in (2) is only used for the disjoint test (1); hence, an artificial bound $\bar{\delta}_i \leq \tau$ can be introduced at any desired location $\bar{\boldsymbol{t}}_i \in D$. We suggest introducing one artificial bound in each corner of D, at the level $\bar{\delta}_i = 0$, since the function $\underline{\delta}$ remains a valid lower bound of g. Higher values of $\bar{\delta}_i$ can be considered to further reduce the risk of sampling too close to the boundary, but then $\underline{\delta}$ might not be a proper lower bound of g, and if a corner corresponds to a collision, more iterations will be needed to find it. Moreover, since τ is often quite small relative to g, using $\bar{\delta}_i = 0$ is a reasonable choice. Note also that with these four artificial bounding points, the minimum of $\underline{\delta}$ is at the center of D; hence, Algorithm 1 remains the same.

A relevant note is that artificial bounds can also be introduced in order to exclude parts of the region D from being considered. This can be useful in cases where the robots is expected to roughly traverse the paths at a predetermined speed, in which case only a diagonal region of D is of interest; see also Section III-D.

If g is known to be cyclic at the boundaries of D, i.e., $g(0,t) = g(T_A, t)$, $g(t, 0) = g(t, T_B)\,\forall t$, then a sample close to a boundary yields information about the opposite boundary; hence, no special action of the initial query point needs to be taken. One typical case in which g is cyclic is when the robot paths themselves are cycles (starting and ending at the same positions).

Another possible way of reducing the boundary effect is to consider larger domains D, i.e., longer robot paths. However, then one might need more information than being the paths disjoint or not: for example, the time of the first collision or the disjoint subintervals of the paths. Such generalizations are available and considered next.

D. Generalization of the Sampling Method

First, assume that the paths considered are sequences of subpaths, consisting of, for example, robot tasks or operations, and some of these subpaths need to be disjoint. Let us suppose the first and second paths consist of m and n subpaths, respectively: One query for each pair of subpaths of interest then generates up to mn queries. However, this would cause a lot of information loss at the boundaries, as discussed in Section III-C. Therefore, one can partition D into the mn subdomains D_{ij}^{sub} corresponding to the subpaths and execute a slightly modified version of Algorithm 1 on D.

Second, finding the first/last points of collision on the paths is sometimes of interest; e.g., in [23] and [28], it is included in the definition of CCD. We denote the first and last collision times by $t_A^{\text{f}}, t_A^{\text{l}}, t_B^{\text{f}}$, and t_B^{l}, where, e.g., t_A^{f} is the first time at which robot A can collide with B, i.e., any $\boldsymbol{t} \in [0, t_A^{\text{f}}) \times [0, T_B]$ is free of collision and, in general, $t_A^{\text{f}} \neq t_B^{\text{f}}$. A common motive for using CA for CCD is that the first collision time is a byproduct; this motive is, however, lost for the efficient binary search version of CA [15] that we generalize here. A naive approach to find $t_A^{\text{f}} \in [0, T_A]$ would repeatedly split the interval in two, using the first half if it contains collisions and the second half, otherwise. However, such an approach could reuse a lot of information between these subintervals; we will show how our sampling strategy can be modified to find the first/last point directly.

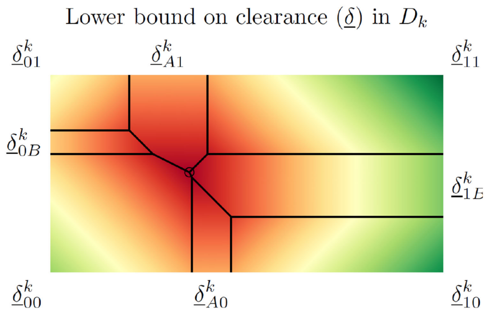

Fig. 3. Illustration of the lower bound (2) in a region D_k, cf., Fig. 2, with the additional lower bounds on the boundaries and the notation used to compute the minimum (5). The black lines denote the intersection of two lower bounding planes and the black circle denotes the obtained minimum $\bar{\boldsymbol{t}}$.

The common feature and difference of these two generalized disjoint queries compared to the original query of disjoint paths (1) is that the sampling should not be aborted once a collision point is found but take some other action. In both cases, i.e., 1) finding the D_{ij}^{sub} that corresponds to disjoint paths, or 2) finding the extreme collision times, we need to mark some rectangular region(s) of D that are no longer of interest. We suggest to use artificial bounds, i.e., in each such rectangular region R define $\underline{\delta}(\boldsymbol{t}) := \tau + \varepsilon, t \in R$. We will describe a procedure that enables this property, but first describe its use to finding the answer to our two other queries. Once a collision is detected, the inequality $\bar{\delta}_N \leq \tau + \varepsilon$ holds and there is a corresponding rhombus region where $g(\boldsymbol{t}) \leq \tau + \varepsilon$ holds. Hence, in case 1), we can exclude all subdomains D_{ij}^{sub} that overlap this rhombus, and in case 2), we can exclude the rectangle $[t_A^{\text{f}}, t_A^{\text{l}}] \times [t_B^{\text{f}}, t_B^{\text{l}}]$ that includes this and previously found rhombuses. Thus, in each iteration, we either (as before) verify a rhombus part of D to be disjoint or extend the subset R of D that is no longer of interest.

The extension of the function $\underline{\delta}$ that is defined by lower bounds in rectangular regions, and not only at points as in definition (2), can be done efficiently. The key insight is that when enforcing a lower bound along a boundary of a region D_k (recall Fig. 2), the lower bounding function $\underline{\delta}$ in the adjacent regions will remain convex. Thus, to introduce a lower bound in a rectangular region R, first, include it as a region of D by splitting some regions D_k, then add the lower bound in corners of R as before, and finally propagate the bounds of boundary lines of R to horizontally and vertically adjacent lines as long as their bounds improve.

The extension also modifies the computation of the minimum of each region D_k [recall (4)]; the solution is still analytical but involves four new terms, namely the bounds along the boundaries of D_k. We denote them as $\underline{\delta}_{A0}^k$, $\underline{\delta}_{A1}^k$, $\underline{\delta}_{0B}^k$, and $\underline{\delta}_{1B}^k$, where, e.g., $\underline{\delta}_{A0}^k$ denotes the bound along the lower boundary of D_k; see Fig. 3. The derivation is as follows: First, assume that the bounds are consistent, i.e., one bound cannot be used to strengthen another, then each bound will define a plane tilting inward into the region. Thus, the minimum value will be at the intersection of three supporting planes and there are two cases: Either the gradients of two planes are linearly dependent and thus create a line where the minimum is achieved [similar to (4)] or there is no pairwise linear dependence and the minimum is achieved in a single point. In the latter case, we have two cases:

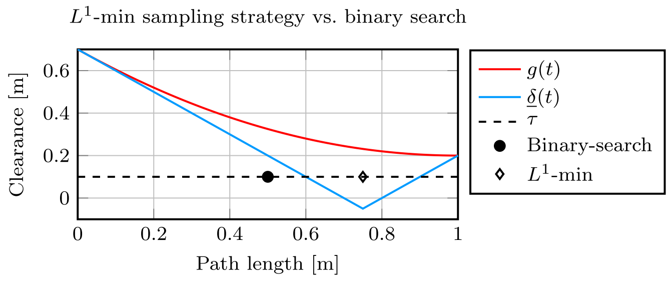

Fig. 4. Illustration of the differences in the sampling strategies for the special case of collision detection among a pair of robot motions; comparison of our suggested method (denoted L^1-min), i.e., minimizing $\underline{\delta}$, with the published CA using binary search, cf., [15].

Either one of the planes corresponds to a side and two planes correspond to corners, or the opposite. However, the consistency assumption implies that the plane corresponding to a side is above the intersection (line) of the two planes corresponding to the corners of the side; hence, this intersection will not correspond to the minimum. To summarize, the minimum in the region D_k is the maximum of eight cases, four of them similar to (4) and defined by two opposing sides or corners, and four defined by two adjacent sides and the opposite corner. The minimum defined by

$$\min_{t\in D_k} \underline{\delta}(\boldsymbol{t}) = \max\left\{ \max_{j\in\{0,1\}} \frac{\underline{\delta}^k_{0j} + \underline{\delta}^k_{1(1-j)} - \Delta T^k_A - \Delta T^k_B}{2}, \right.$$
$$\frac{\underline{\delta}^k_{A0} + \underline{\delta}^k_{A1} - \Delta T^k_B}{2}, \frac{\underline{\delta}^k_{0B} + \underline{\delta}^k_{1B} - \Delta T^k_A}{2},$$
$$\left. \max_{i,j\in\{0,1\}} \frac{\underline{\delta}^k_{ij} + \underline{\delta}^k_{A(1-j)} + \underline{\delta}^k_{(1-i)B} - \Delta T^k_A - \Delta T^k_B}{3} \right\} \tag{5}$$

replaces (4) as priority in the queue.

E. Specializations of the Sampling Method

The focus of this article has been to determine if two robot paths are disjoint. However, the L^1-min sampling strategy can be specialized to the case when one of the robots is a static environment. The domain D then becomes a line and thus the data structure for optimizing $\underline{\delta}(\boldsymbol{t})$ is simplified accordingly. Similarly, if two robot motions are to be checked for collision, i.e., that $g(t,t) \leq \tau$ for some $t \in [0,T]$, then this corresponds to analyzing a path $\boldsymbol{\phi}(t)$ in the domain D. Moreover, by adding the two robots' velocities along the path $\boldsymbol{\phi}(t)$, the collision test becomes essentially same as in the static environment case.

The resulting sampling strategy is a version of the binary strategy commonly used within CA; cf., [15]. The difference is that our method utilizes more information from previous samples to derive the position of the next sample point. The effect is that the next sample point is not centered between previous samples; it is instead centered in the largest interval that is not yet determined as collision-free; see Fig. 4. Note that in this example, our sampling method completes after one iteration while the binary search requires two iterations.

Finally, in the light of this resemblance, the L^1-min sampling is a variant of CA that verifies a pair of paths to be disjoint, while utilizing all computed clearance information.

F. Robot Decomposition

There is another significant difference to the binary sampling strategy suggested in [15]: This approach decomposes the robot into one part per link. By doing this, they retrieve more detailed information: For example, a slow-moving part can easily be verified disjoint even though the clearance is low. Moreover, their binary search strategy ensures that the parts are still evaluated at the same position of the motion, which allows for efficient caching of the kinematic results.

In our framework, this decomposition would result in one lower bounding function $\underline{\delta}$ for each pair of links such that the next sample equals the minimum of all these functions and g is evaluated for each pair of links (i.e., a previously partial result). However, for our instances (cf., Section V-A), preliminary results showed little to no gain in the lower bound as well as in the number of clearance evaluations required. This is because, for our instances, it is very frequent that the tools (attached to the robot faceplate) have both the greatest velocities and the lowest clearances.

Not decomposing the robot also has benefits when it comes to evaluating g. In fact, we only need to compute the exact distance between pairs of parts if their distance is smaller than for any previously computed pair. This aspect is described more in detail in Section III-G.

G. Greedy Clearance Computation

Our L^1-min sampling strategy does not require that the clearance function $g(\bar{\boldsymbol{t}}_N)$ is computed (line 7) if $g(\bar{\boldsymbol{t}}_N) > \tau$, in which case any $\bar{\delta}_N$ such that $\tau < \bar{\delta}_N \leq g(\bar{\boldsymbol{t}}_N)$ suffices. This has a great benefit, as computing the clearance between two rigid bodies using BVH allows for early termination, i.e., if $g(\bar{\boldsymbol{t}}_N) > \tau$, then it holds that $g(\bar{\boldsymbol{t}}_N) \geq \bar{\delta}_N > \tau$, but if $g(\bar{\boldsymbol{t}}_N) \leq \tau$, then the computation is carried out exactly and $\bar{\delta}_N := g(\bar{\boldsymbol{t}}_N)$. However, here a tradeoff occurs; a large value of $\bar{\delta}_N$ maximizes the excluded region, while a small value enables a faster clearance computation. Here, we use the strategy of computing g inexactly when there is risk of redundant information; hence, we use $\tau + \min\{\tau - \underline{\delta}(\bar{\boldsymbol{t}}_N), \tau_{\mathtt{max}}\}$ as a break point (note that $\underline{\delta}(\bar{\boldsymbol{t}}_N)$ is already computed, on line 13). The term $\tau - \underline{\delta}(\bar{\boldsymbol{t}}_N)$ "mirrors" the lower bound in the threshold level τ; hence, computing a distance larger than this term will cause overlaps with regions known to be disjoint. The term $\tau_{\mathtt{max}}$ is as a heuristic improvement that applies in early iterations before the lower bounding function $\underline{\delta}$ becomes tight enough. For our instances (cf., Section V-A), $\tau_{\mathtt{max}} := 0.1$ m is a reasonable choice.

One advantage of using the lower bounding function $\underline{\delta}$ to determine the precision of the clearance computation is that a tight bound yields a faster evaluation of g. Preliminary results indicated that the last iterations were several times faster than the first ones and that equally much time was spent on the first 30% and last 70% of the iterations, respectively. This is despite the observed complexity of optimizing $\underline{\delta}$ (Section III-B). This relates to the discussion (see Section III-B) on large overlaps for the artificial clearance function $g = \tau + c$ that resulted in a rather large number of iterations. However, as the majority of those iterations had a tight lower bound, it can be argued that these overlaps are not an issue but could even be beneficial in terms of computation time when g is much cheaper to approximate than to compute.

Moreover, note that these kinds of thresholds can be used to accelerate clearance routines other than those based on BVH; see, e.g., [43] and [44]. Hence, such routines could also be used in conjunction with our sampling strategy.

IV. Swept Volume Generation and Distance Test

An alternative approach to verify that two paths are disjoint [see (1)] is to check if the swept volumes of the paths are separated by the desired threshold. This could be done implicitly by using a conservative BVH approach, such as OBB or LSS; however, since most BVH approaches depend on higher-order information, cf., [28], we consider this an unfair comparison and outside the scope of this contribution, recall Section I-B. Moreover, it might be beneficial to build the swept volumes explicitly (with a certain precision) to enable their reutilization in later queries.

One way to find out when explicit sweeps might be beneficial is to consider the case when two large sets of M independent paths each are to be checked for pairwise collision. Then the number of tests is $\mathcal{O}(M^2)$, whence the aim is to reduce the computational effort spent in each test: Generating the sweeps is an $\mathcal{O}(M)$ operation, which is computationally cheaper whenever M is sufficiently large. This section challenges the method in Section III, and Section V finds the breaking point at which the swept volume approach becomes preferable.

There are many ways to implement a swept volume generation (cf., [32]–[34], [45]); however, one needs to decide whether it needs to be exact or if an approximation is good enough. Our initial attempt aimed at using an exact swept volume algorithm (tracing the triangles). Preliminary tests resulted, however, in an unreasonably large value of M. Therefore, we focused on octree approximations, which were roughly a hundred times faster to compute, using an octree depth of eight. Moreover, this choice was also motivated by the fact that checking if two aligned octrees overlap is a very cheap operation and can be neglected in the analysis of M. An exact approach, on the other hand, would require computing the distance between two triangular meshes. Likewise, since the goal is to evaluate our suggested sampling strategy (Algorithm 1), we decided that a straightforward approach is the most suitable.

To generate an octree representing the swept volume, we utilize a spherical cover of each link of the robot, i.e., spheres with radii τ_{p} covering the links. In our work, we ensure that $\tau_{\mathrm{p}} \leq \frac{D^{\mathrm{cell}}(d)}{4}$, i.e., a quarter of the leaf octree cell diameter of the given octree depth d. Moreover, the path is sampled such that the distance between two consecutive samples is less than $2\tau_{\mathrm{p}}$ (thus, the computing time scales linearly with the path length). Thus, to ensure that the entire sweep is contained in the octree, spheres with radius $2\tau_{\mathrm{p}}$ are inserted for each sphere and path sample. Note that, when there is a clearance demand in (1), i.e., $\tau > 0$, τ_{p} remains unchanged but the cover of spheres is generated to enclose the links padded with $\tau/2$. The result is that the octree intersection test becomes conservative, i.e., if two octrees are disjoint, then so are the corresponding paths, whereas the reversed implication does not hold.

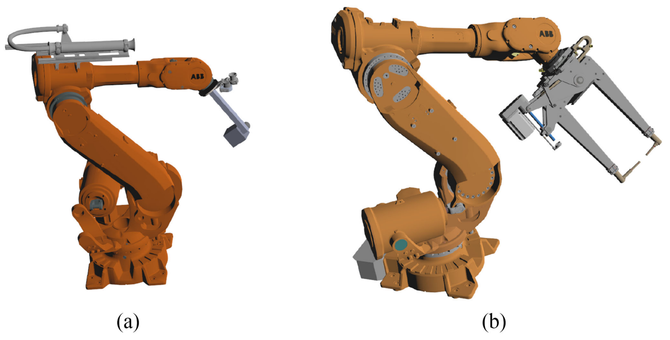
Fig. 5. Illustration of the robot and tool geometries used in our instances, apart from the geometry visualized in Fig. 1. (a) Cam. (b) Spot.

V. Results

Our computational experiments were conducted on a computer with a 3.5-GHz Intel i7-4770 K CPU, implemented in C++, and to simplify the analysis, all algorithms run on a single thread, and, thus, the computing time and CPU time are essentially the same.

A. Test Instances

Our test instances are from three industrial scenarios corresponding to three types of robot tools: 1) a stud weld tool (cf., Fig. 1), 2) a sensor tool [cf., Fig. 5(a)], and 3) a spot weld tool [cf., Fig. 5(b)]. The number of triangles (robot and tool) were 92 627, 360 270, and 642 125, respectively. In all cases, the robots are situated in a working cell consisting of four robots with some tasks to perform on an assembly workpiece. For each scenario, a load balanced solution was computed, resulting in several robot paths for each task sequence. Normally, a fixed-path coordination scheme (cf., [8]) prevents robot–robot collisions by utilizing that such paths are disjoint.

To retrieve more paths of greater variety, we performed the following operations. First, since these paths are naturally rather short (originating from optimized sequences), they can be merged, resulting in a longer path. Second, by perturbing the location of the tasks, other sequences, and associated paths, were found and included in the test set. Overall, at least 300 paths were generated and analyzed for each tool; this is in order to ensure sufficiently many samples when generating the octrees, recall Section IV, and sufficiently many path pairs with a large enough clearance for verifying them disjoint—recall Algorithm 1.

In our applications, we use $\tau = 0.1$ m, whereas the robots are a few meters long. For the unit-velocity parametrization, a conservative adaptive sampling technique was used; see Section I-B for other alternatives. The resulting path lengths vary from 0.01 to 4 m; most are, however, around 0.2 m.

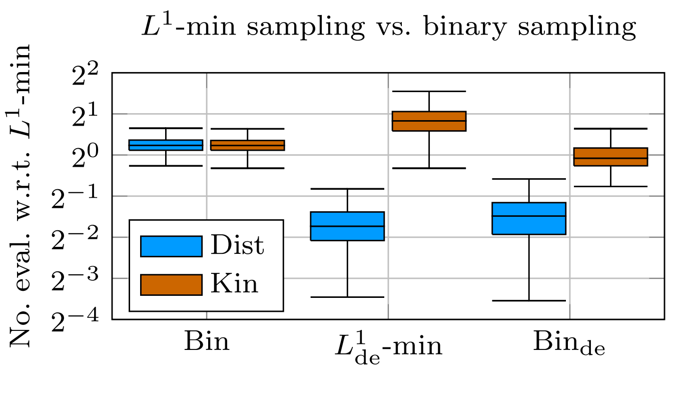

Fig. 6. Binary sampling strategy without the robot decomposition (Bin), its decomposed version (Bin_{de}), and the decomposed version of L^1-min sampling strategy (L^1_{de}-min), all relative to the L^1-min sampling strategy. The statistics compared are number of distance computations between links (Dist) and the number of kinematic evaluations (Kin). For each statistic and considered motion pair, a score is computed as the ratio between one of the three strategies and the L^1-min strategy, and the boxplots illustrate resulting distribution.

B. Special Case of Motions

We now consider the special case of detecting collisions among pairs of motions; cf., Section III-E. The motions are retrieved from the paths by using the robots' maximum velocities. We compare the L^1-min and the binary sampling strategies by the number of kinematic and distance evaluations used. We consider two variations of each strategy, with and without decomposing the robots into one part per link; cf., Section III-F.

The results are illustrated in Fig. 6. The L^1-min sampling strategy uses roughly 15% fewer kinematic and distance computations compared to a variant of the binary sampling that do not decompose the robot (Bin). When decomposing the robot into one part per link, we see similar trends for both the Bin_{de} and the L^1_{de}-min strategies, i.e., much fewer distance computations (the computational effort is, however, only 20% smaller; cf., Section III-G). Moreover, the kinematic caching used in Bin_{de} results in almost half as many kinematic evaluations compared to L^1_{de}-min, which, on the other hand, use 12% fewer distance computations on average. Thus, for this special case, our sampling method provides a competitive and comparable alternative to existing algorithms.

C. Time to Verify Disjointness

In order to compare our suggested method for verifying two path disjoints (Algorithm 1) with the basic approach of approximating the swept volumes with an octree, we begin by presenting the results for the latter. This will then be used to determine how many paths (M) that are required before the swept volume approach is preferable to Algorithm 1. Note, however, that the swept volume approach presented here is conservative; hence, it cannot deduce that two paths intersect but only that they are disjoint or that they are close and might intersect. But as the precision, or maximum depth, of the octree increases, the results become more reliable (as the cell sizes decrease).

Fig. 7 presents the CPU time for generating an octree of the swept volume of a path divided by the length of the path, for three different robot tools, and four different depths of the octree. In

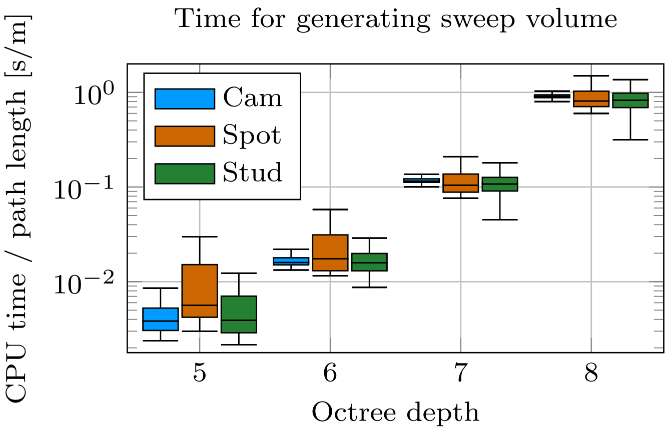

Fig. 7. CPU time of generating sweep volume approximation for three different tools (camera, spot-weld, and stud-weld) using four different precisions (octree depths). The domain of the octree was always a $8\tilde{~}\text{m}^3$ cube, and, thus, the cell sizes are in the interval $[2^{-5}, 2^{-2}]\ \text{m}^3$. For each tool and depth, at least 300 sweeps were approximated and the boxplot of the resulting CPU time per path length [T_A or T_B in (1)] is presented; recall Section IV motivating this linear relation.

Section IV, it was deduced that there is a linear relation between the path length and the CPU time (and the number of inserted spheres). Hence, their ratio is presented. For example, a sequence typically has a few meters path length; approximating its sweep with an octree using low precision (0.25 m) is computed within hundreds of a second, whereas a higher precision (0.03 m) may require a second.

Note that there is a spread in the linear coefficient, i.e., the CPU time is determined not only by the length of the path. Other properties, such that the total volume or surface area of the swept volume, may correlate better with the CPU time. One reason can be that the path length is defined by the maximum velocity (typically the velocity of the tool) which typically represents only a small part of the swept volume. This results in spheres being inserted in already allocated octree cells, which is faster than allocating a new cell.

The main results concerning the CPU time of our L^1-min sampling strategy (Algorithm 1) are displayed in Fig. 8(a). To enable a comparison with the octree sweep approximation, the minimum clearances are grouped into bins corresponding to octree depth. Thus, all path pairs in a bin can be determined disjoint by the octree approximation of the corresponding depth. The first four bins correspond to the depths reported in Fig. 7 and the last bin to instances that required deeper (possibly infinitely deep) octrees.

The CPU time is normalized here by the so-called path area, i.e., the product of the two path lengths. This is reasonable since the CPU time is roughly proportional to the number of clearance computations (recall Section III-A), which, in turn, determines an area free of collision points (cf., Fig. 2). However, the results presented in Fig. 8(a) shows that there is greater spread within each bin as compared to Fig. 7. This is, first, since the excluded area depends on the clearance, but a path pair with only a small region with low clearance can be verified faster than a pair with low clearance in a large region. And second, since the clearance computations are faster for large clearances; cf., Section III-G.

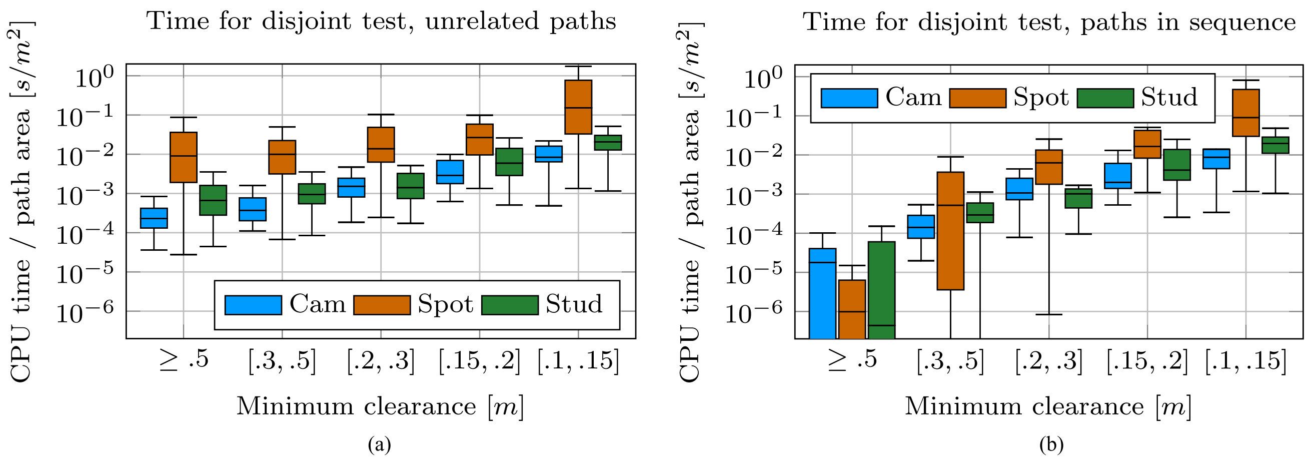

Fig. 8. CPU time of verifying robot pairs of paths disjoint, for three different tools (camera, spot-weld, and stud-weld) and for five different clearances of the swept paths. These five bins correspond to the sum of the diagonal of an octree cell and the tolerance τ, cf., Fig. 7, to enable comparing the two approaches. Each tool and clearance bin comprise 20–400 pairs of paths. The resulting boxplot for the ratio of the CPU time and the path area $T_A T_B$ is presented. (a) Unrelated paths are assumed and Algorithm 1 is used. (b) Paths are assumed to be part of sequences and the generalization of Algorithm 1 from Section III-D is used.

These two effects contribute to the observed spread within each bin.

Since the paths in our test instances originate from sequences of paths, we can utilize the generalization of Algorithm 1 described in Section III-D. Recall that this generalization reuses the information from previous queries. The result is presented in Fig. 8(b), which shows several factors of reduction in CPU time compared to not reusing the information [as in Fig. 8(a)].

The reason for the heavy lower tails in the boxes of Fig. 8(b) is that a pair of paths with high clearances can be determined disjoint using clearance computations from neighboring paths in the sequence. Hence, in this data, there are lots of zeroes for shorter paths with high clearance. Moreover, the axis is cut off at $2 \cdot 10^{-7}$ to have the same scale as Fig. 8(a) and since all measurements below this correspond to zeroes.

Also, by utilizing that our paths are parts of sequences, we also noted that the reduction in the number of clearance computations was even larger than the reduction in CPU time. This may be due to that most (including our) clearance computations are warm started; hence, good locality (in the sample set) can speed up computations. Thus, there is room for improvement considering long paths; see Section VI-A.

Comparing Figs. 7 and 8(a), we may investigate the number of paths of a certain length that are needed before the sweep octree approximation becomes a useful filter. Assume M paths of similar length $\approx l$ for each robot. Let $t_{\mathrm{p}}(c)$ denote the expected CPU time per path area to verify a path pair with minimum clearance c and $t_{\circ}(c)$ denote the expected CPU time per path length to generate a sweep with octree depth sufficient for c. A preliminary result showed that the CPU time of the octree intersections are magnitudes faster than $t_{\mathrm{p}}(c)$. We yield that when $t_{\mathrm{p}}(c)l^{2\,\tilde{}}M^2 \geq 2t_{\circ}(c)lM$, the octree is a useful filter. This reduces to

$$M \geq \frac{2t_{\circ}(c)}{lt_{\mathrm{p}}(c)} \qquad (6)$$

which is used together with the results presented in Figs. 7 and 8(a) to retrieve Fig. 9. Moreover, note that the relation (6) is a very simplistic model and is not intended to be used for finding the exact breaking point but rather to visualize trends and magnitudes. For example, the neglected CPU time for the octree intersection test might be significant for deep octrees.

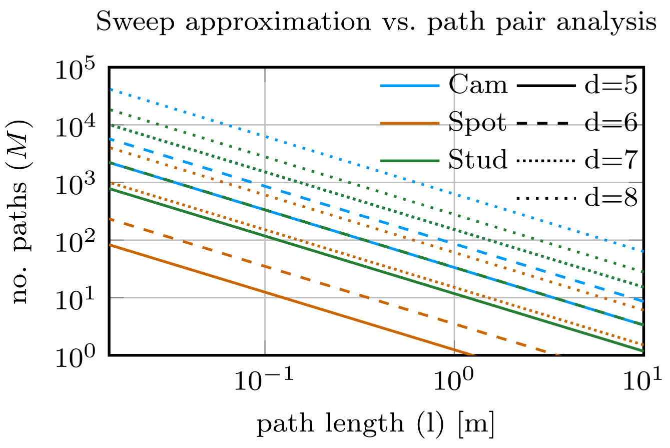

Fig. 9. Illustration of the breaking point (6), i.e., when the sweep approximation becomes faster than Algorithm 1 to verify disjointness. This is in terms of the number of paths per robot and their length. Here, d is the octree depth, and $t_{\mathrm{p}}(c)$ and $t_{\circ}(c)$ are taken to be the median in the respective bins. Moreover, $t_{\mathrm{p}}(c)$ is computed by not assuming the paths that are part of a sequence, i.e., from Fig. 8(a).

The figure shows that the breaking point at which filtering using the sweep approximation becomes beneficial depends on both the robot type and the minimum clearance of the paths. Hence, if it is known beforehand that the paths are likely to have a high clearance, then using a shallow octree is beneficial. However, only for very long or numerous paths should a deep octree be applied. The sequences from our test case contain mostly short paths (movements between adjacent tasks) and only a few longer ones. Hence, using the octree sweep approximation at any depth will not speed up computations much, if at all. Moreover, since the paths are part of sequences, as illustrated in Fig. 8(b), the generalized sampling strategy is significantly faster for high clearances. As a result, all lines in Fig. 9 will roughly pass through $(l, M) = (10^{-2}, 10^5)$ by using the sequence assumption. Thus, even long paths need to be numerous before there is any gain in applying the octree approach; e.g., ($l \approx 1$ m) requires thousands of paths and thus millions of pairs.

VI. Conclusion

In this article, we proposed a method for minimizing a bivariate function with a known L^1 Lipschitz constant and utilized it to determine if a pair of robot paths is disjoint, i.e., if their respective swept volumes are disjoint. Our method is related to the traditional CA techniques that determine if a pair of robot motions collide; our method can be specialized to check this and is then similar to CA using binary sampling.

We showed that even though the sampling strategy involves solving a bivariate L^1 Lipschitz optimization problem, the introduced overhead is small compared to the time of positioning the robots and computing clearances. Moreover, we showed that this will remain true even when a very large number of sample points is required.

One main observation made is if the paths are parts of longer paths, then there is great gain in modifying the sampling strategy to consider two longer paths and simply record all found colliding pairs of paths. This is because a lot of clearance information is discarded by considering all pairs of paths separately. This is of great importance since it is often more informational to know the regions of configurations that are collision-free compared to an existence certificate.

Finally, we compared our sampling strategy with the intersection test of octrees that represent the swept volumes of the robot paths: We noticed that the octrees would seldom be of use, unless there are many pairs of long paths that are well-separated. Moreover, when considering subpaths, our sampling strategy performed much better, hence verifying the paths disjoint using octrees are not useful in such situations.

A. Further Work

In the framework of smart assembly, we can utilize this new collision query to efficiently detect critical positions of the robot paths, which can be used by motion planning algorithms.

We suggest four potential improvements of the sampling strategy. First, great improvement potentials rely on the parallelization of the search, regarding several distributed points and not only the global minimum. Second, the possibility to localize a minimum point, e.g., preferring points close to previous points, to get better bounds that accelerate the distance computations or to store some data structure for warm starting the clearance computation based on previous results. Third, using directional information and not only maximum velocity along the respective robot path. This could be done within the framework of a volume bounding hierarchy; cf., [41]. However, it would require additional computational effort since not only distance but also direction between the bounding volumes would need to be computed. Moreover, since the shortest distance directions depend on the positions of both robots, the clearance bound would not be directly dependent on the L^1 distance in the time domain. Fourth, consider using higher-order information to construct BVH based on Taylor expansion, cf., [28], should provide a better filter than our direct octree approach provided that the higher-order information is available and a special BVH traversal for the disjoint test is developed.

Finally, we like to apply our sampling algorithm to other applications, e.g., another robot type, mobile automated guided vehicles or other bivariate functions with known L^1 Lipschitz constants.

Acknowledgment

Finally, we would like to thank the anonymous reviewers for their valuable feedback.

References

[1] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proc. ICRA. Millennium Conf. Int. Conf. Robot. Autom.*, vol. 1, 2000, pp. 521–528.

[2] L. Palmieri, L. Bruns, M. Meurer, and K. O. Arras, "Dispertio: Optimal sampling for safe deterministic motion planning," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 362–368, Apr. 2020.

[3] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Comput. Sci. Dept., Iowa State University, Ames, IA, USA, Tech. Rep. 98–11, 1998.

[4] A. Agha-Mohammadi, S. Agarwal, S. Kim, S. Chakravorty, and N. M. Amato, "SLAP: Simultaneous localization and planning under uncertainty via dynamic replanning in belief space," *IEEE Trans. Robot.*, vol. 34, no. 5, pp. 1195–1214, Oct. 2018.

[5] M. da Silva Arantes, C. F. M. Toledo, B. C. Williams, and M. Ono, "Collision-free encoding for chance-constrained nonconvex path planning," *IEEE Trans. Robot.*, vol. 35, no. 2, pp. 433–448, Apr. 2019.

[6] P. A. O'Donnell and T. Lozano-Pérez, "Deadlock-free and collision-free coordination of two robot manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 1, 1989, pp. 484–489.

[7] S. M. LaValle, "Hierarchical methods," in *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, pp. 210–212.

[8] D. Spensieri, R. Bohlin, and J. S. Carlson, "Coordination of robot paths for cycle time minimization," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2013, pp. 522–527.

[9] D. Hömberg, C. Landry, M. Skutella, and W. A. Welz, "Automatic reconfiguration of robotic welding cells," in *Math for the Digital Factory*. Berlin, Germany: Springer, 2017, pp. 183–203.

[10] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 856–869, Aug. 2018.

[11] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Trajectory planning in robotics," *Math. Comput. Sci.*, vol. 6, no. 3, pp. 269–279, 2012.

[12] M. Ragaglia, A. M. Zanchettin, and P. Rocco, "Trajectory generation algorithm for safe human-robot collaboration based on multiple depth sensor measurements," *Mechatronics*, vol. 55, pp. 267–281, 2018.

[13] R. Söderberg, K. Wärmefjord, J. S. Carlson, and L. Lindkvist, "Toward a digital twin for real-time geometry assurance in individualized production," *Corporate Insolvency Resolution Process Ann.*, vol. 66, no. 1, pp. 137–140, 2017.

[14] R. Culley and K. Kempf, "Collision detection algorithm based on velocity and distance bounds," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 3., 1986, pp. 1064–1069.

[15] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking of robot paths," in *Algorithmic Foundations of Robot V*. Berlin, Germany: Springer, 2004, pp. 25–41.

[16] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[17] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1163–1177, Oct. 2016.

[18] E. Åblad, "Intersection-free load balancing for industrial robots," Master's thesis, Dept. Math. Sci., Chalmers Univ. Technol. Univ. Gothenburg, Gothenburg, Sweden, 2016.

[19] E. Åblad, D. Spensieri, R. Bohlin, and J. S. Carlson, "Intersection-free geometrical partitioning of multirobot stations for cycle time optimization," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 842–851, Apr. 2018.

[20] E. Åblad, A.-B. Strömberg, and D. Spensieri, "Mathematical modelling for load balancing and minimization of coordination losses in multirobot stations," Licentiate thesis, Dept. Math. Sci., Chalmers Univ. Technol. Univ. Gothenburg, 2020.

[21] F. Imeson and S. L. Smith, "An SMT-based approach to motion planning for multiple robots with complex constraints," *IEEE Trans. Robot.*, vol. 35, no. 3, pp. 669–684, Jun. 2019.

[22] S. Redon, M. C. Lin, D. Manocha, and Y. J. Kim, "Fast continuous collision detection for articulated models," *J. Comput. Inf. Sci. Eng.*, vol. 5, no. 2, pp. 126–137, 2005.

[23] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," in *Computer Graphics Forum*, vol. 21, no. 3. Wiley Online Library, 2002, pp. 279–287.

[24] P. M. Hubbard, "Collision detection for interactive graphics applications," *IEEE Trans. Vis. Comput. Graphics*, vol. 1, no. 3, pp. 218–230, Sep. 1995.

[25] R. Geraerts and M. H. Overmars, "Sampling techniques for probabilistic roadmap planners," in *Proc. Int. Conf. Intell. Auton. Syst.*, 2004, pp. 600–609.

[26] G. Sánchez and J.-C. Latombe, "Single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Robotics Research*. Berlin: Springer, 2003, pp. 403–417.

[27] X. Xu, F. Sun, Y. Luo, and Y. Xu, "Collision-free path planning of tensegrity structures," *J. Struct. Eng.*, vol. 140, no. 4, pp. 1–9, 2013.

[28] X. Zhang, S. Redon, M. Lee, and Y. J. Kim, "Continuous collision detection for articulated models using taylor models and temporal culling," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 15–es, 2007.

[29] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates," *Int. J. Robot. Res.*, vol. 35, no. 7, pp. 767–796, 2016.

[30] B. Lacevic, D. Osmankovic, and A. Ademovic, "Path planning using adaptive burs of free configuration space," in *Proc. XXVI Int. Conf. Inf., Commun. Autom. Technol., Sarajevo, Bosnia-Herzegovina*, 2017, pp. 1–6.

[31] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, 1998.

[32] A. Foisy and V. Hayward, "A safe swept volume method for collision detection," in *Proc. 6th Int. Symp. Robot. Res.,*1994, pp. 62–68.

[33] M. Herman, "Fast, three-dimensional, collision-free motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 3, 1986, pp. 1056–1063.

[34] N. Ahuja, R. T. Chien, R. Yen, and N. Bridwell, "Interference detection and collision avoidance among three dimensional objects," in *Proc. 1st AAAI Conf. Artif. Intell.*, 1980, pp. 44–48.

[35] R. H. Mladineo, "An algorithm for finding the global maximum of a multimodal, multivariate function," *Math. Program.*, vol. 34, no. 2, pp. 188–200, 1986.

[36] S. Jakobsson, P. Lindroth, and A.-B. Strömberg, "A minimax strategy for global optimization," 2011. [Online]. Available: https://research.chalmers.se/publication/140359

[37] Y. D. Sergeyev and D. E. Kvasov, "Lipschitz global optimization," in Deterministic Global Optimization. New York, NY, USA: Springer, 2017, pp. 1–17.

[38] F. Aurenhammer and R. Klein, "Voronoi diagrams," in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. New York, NY, USA: Elsevier, 2000, pp. 201–290.

[39] M. Jünger, V. Kaibel, and S. Thienel, "Computing Delaunay triangulations in Manhattan and maximum metric," *Institut für Informatik*, Universität zu Köln, Tech. Rep. 94.124, 1995.

[40] P. Jiménez, F. Thomas, and C. Torras, "Collision detection algorithms for motion planning," in *Robot Motion Planning and Control*. Berlin, Germany: Springer, 1998, pp. 305–343.

[41] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast distance queries with rectangular swept sphere volumes," in *Proc. IEEE Int. Conf. Robot. Autom. Millennium Conf.*, vol. 4., 2000, pp. 3719–3726.

[42] *Gurobi Optimization*, LLC, "Gurobi optimizer reference manual," 2019. [Online]. Available: http://www.gurobi.com

[43] D. Schneider, E. Schömer, and N. Wolpert, "Collision detection for 3D rigid body motion planning with narrow passages," in *Proc. IEEE Int. Conf. Robot. Autom.,* Singapore, 2017, pp. 4365–4370.

[44] M.Ghosh, S.Thomas, and N. M.Amato, "Fast collision detection for motion planning using shape primitive skeletons," in *Algorithmic Foundations of Robot. XIII, Ser. Springer Proc. Advanced Robot.*, vol. 14., M. Morales, L. Tapia, G. Sánchez-Ante, and S. Hutchinson, Eds. Cham, Switzerland: Springer, 2018, pp. 36–51.

[45] K. Abdel-Malek, J. Yang, D. Blackmore, and K. Joy, "Swept volumes: Foundation, perspectives, and applications," *Int. J. Shape Model.*, vol. 12, no. 1, pp. 87–127, 2006.

Edvin Åblad was born in 1991. He received the M.S. degree in engineering mathematics and computational science from the Chalmers University of Technology, Gothenburg, Sweden, in 2016. His M.S. thesis was on intersection-free load balancing for industrial robots. He is currently working toward the Ph.D. degree in mathematics on load balancing multirobot stations at the Geometry and Motion Planning Department, Fraunhofer-Chalmers Research Centre for Industrial Mathematics and the Department of Mathematical Sciences, Chalmers University of Technology.

His current research interests include computational geometry, multiagent modeling, and optimization algorithms.

Domenico Spensieri was born in 1978. He received the "Laurea" degree in computer engineering and control systems from the University of Pisa, Pisa, Italy, in 2003, and the Licentiate degree in product and production development from the Chalmers University of Technology, Gothenburg, Sweden, in 2017. He is currently a Ph.D. candidate with the Department of Industrial and Materials Science working on product and production development.

He joined the Fraunhofer-Chalmers Centre, Gothenburg, Sweden, in 2004, where he works with the Geometry and Motion Planning Group as an Applied Researcher and System Developer. His current main research interests include multiagent optimization, assembly planning, and robotics.

Robert Bohlin was born in 1972. He received the Ph.D. degree in mathematics on robot path planning from Chalmers University of Technology, Gothenburg, Sweden, in 2002.

He is a Researcher and Project Manager with the Geometry and Motion Planning Department, Fraunhofer-Chalmers Research Centre for Industrial Mathematics (FCC), Gothenburg. His research interests include methods, algorithms, and tools for virtual product realization and in particular automatic path planning, collision detection, simulation of flexible material, optimization, and kinematics.

Ann-Brith Strömberg was born in 1961. She received the Ph.D. degree in mathematics on subgradient methods from Linköping University, Linköping, Sweden, in 1997.

She is a Professor of Mathematical Optimization with the Department of Mathematical Sciences, Chalmers University of Technology, Gothenburg, Sweden, and the University of Gothenburg, Gothenburg. Her research concerns mathematical modeling and solution of optimization problems, including discrete, convex nonsmooth, simulation based, and multiobjective optimization. Much of her research is carried out in cooperation with academy and industry and includes scheduling of production and maintenance, load balancing, integration of variable electricity generation in the energy system, and transport planning.