Stress Testing of Single-Arm Robots Through Constraint-Based Generation of Continuous Trajectories

Mathieu Collet*, Arnaud Gotlieb*, Nadjib Lazaar† and Morten Mossige ‡
*Simula Research Laboratory, Oslo, Norway
Email: {mathieu, arnaud}@simula.no
†LIRMM, University of Montpellier, CNRS, Montpellier, France
Email: lazaar@lirmm.fr
‡ABB Robotics, Bryne, Norway
Email: morten.mossige@no.abb.com

***Abstract*—System Testing of Single-Arm Robots (SAR) is challenging as typical SAR involve multiple coordinated software-controlled subsystems, such as motion and action control, perception and anti-collision systems. Developing convincing test scenarios which place the SAR into highly CPU-demanding cases is complicated due to the huge number of possible robots' workspace configurations. In this paper, we introduces *RobTest*, a tool-supported method for stress testing of SAR, which generates automatically optimal collision-free trajectories. Initially specified by a cloud of points and a set of obstacles, these trajectories are piecewise linear paths in a cost-labelled oriented graph. By using advanced Constraint Programming (CP) techniques, such as constraint refutation over continuous domains and constraint optimization over graphs, RobTest can generate continuous trajectories which 1) avoid physical obstacles and 2) maximize the load of the various CPUs of the SAR. These trajectories result into automatically robot computer programs which place the SAR into high-demanding test scenarios. Our initial experimental evaluation of *RobTest* shows promising results.**

I. Introduction

Single-Arm industrial Robots (SAR) are complex cyber-physical systems which need to be thoroughly tested before being deployed. System testing of SAR involve various processes and among them, checking if specified trajectories are actually followed up by the robot is a crucial challenge. Discrepancies between the specified path and the actual path may occur for several reasons due to either mechatronic engineering problems, or robot fatigue or control software issues. Revealing such discrepancies earlier in the test process is crucial to 1) reduce the maintenance costs, as discovering a bug on already deployed robots entail expensive robot repair costs at customer site, 2) to reveal any changes in the implementation that can lead to critical issues for the customer.

Formally speaking, given an initial cloud of points in a 3D-space, that can be hitted by the robot operating nose, the test objective (**T_Obj**) consists in finding collision-free and loop-free trajectories between these points which 1) avoid all the possible obstacles of the robot workspace and 2) maximize the CPU load of the robot. When found, these trajectories can be converted into computer programs for the robot and used as test scenarios. Maximizing the CPU load is crucial to select error-prone test scenarios positionning the SAR into stressing conditions.

Solving this challenging problem (**T_Obj**) is part of the general area of robot motion planning [1], but it must be distinguished from the *optimal planning of robot trajectories* problem [2], where the goal is to generate collision-free trajectories which minimize the time for the SAR to perform its task. Unlike **T_Obj** , this problem entails solving complex equations over the reals related to the kinematics of the robot. In industrial settings, **T_Obj** is currently tackled by test engineers without any automated support. Thus, only clouds with a small number of points are considered (typically 4-5 points) and only a few trajectories are considered (typically a single trajectory). Nevertheless, some research results have been reached to deal with related problems. One of the first proposed method in the context of trajectory generation has considered robot's workspace discretization [3]. The idea is to divide the workspace into equivalent subboxes and discard boxes which contain obstacles. Then, by using an exploration mechanism which looks for neighbour boxes and advanced results of Interval Analysis, the method generates a continuous trajectory. The method produces trajectories with guaranteed computations over the reals [4], [5] which is an important and difficult problem. In our work, we also considered equations and inequations over the reals but we do not claim any guarantee on the absence of rounding errors. With the discretization of the robot workspace, it is possible to generate collision-free trajectories but generating cost-optimal trajectories still require deeper exploration. In our work we focussed on the optimization problem. Another method considers the so-called *joint space* to model robot trajectories [6], [7]. The joint space includes all the robot'skinematics and can determine precisely the position reached by the robots. This is interesting for determining the so-called singularity problems and provide alternatives to avoid them [8]. However, the computational time required by these methods is usually prohibitive for inclusion into a continuous integration process. Another method

considers planning and re-evaluation of trajectories. The goal is to plan different trajectories in advance and re-plan the current trajectory when a specific unwanted configuration is reached [9]. This method is original as it considers dynamic trajectory generation but it cannot be considered for testing purposes as it dynamically re-assign tasks to the robot.

In this paper, we introduce *RobTest*, a tool-supported method for stress testing of SAR, which generates automatically collision-free and loop-free cost-maximized trajectories. These trajectories are generated with advanced Constraint Programming (CP) techniques such as constraint refutation over continuous domains and constraint optimization over graphs. Constraint refutation allows *RobTest* to determine which pair of points is reachable or not, while constraint optimization over graphs allows effective search-space prunings of search tree. This latter process enables the maximization of a cost function which captures the notion of CPU load. *RobTest* can generate loop-free trajectories which 1) avoid physical obstacles and 2) maximize the load of the various CPUs of the SAR. This paper reports on an experimental evaluation of *RobTest* over initial cloud of points containing up to 19 points. Our initial experimental evaluation of *RobTest* shows promising results.

The rest of the paper is organized as follows. Sec. 2 presents the general background of the approach with its notations and the formalization of the problem. Sec. 3 details *RobTest* with its main components and the usage of advanced Constraint Programming methods. Sec. 4 presents our experimental evaluation and examines in depth 3 research questions. Finally, Sec. 5 concludes the paper and draws a couple of perspectives.

II. Background

Stress testing the motion control of a given SAR requires to define test scenario where the robot performs a succession of motions at various speeds and measuring the value of some metrics. The starting point of the process is given by the tester under the form of a cloud of points in a 3D-space. In the rest of the section, we describe first the robot working space, using 2D-projections for the sake of simplicity. Then, we explain the possible robot motions and how they are usually encoded. The concept of trajectories is then presented and the section is ended by a brief description of the current stress testing process.

A. Notations

Given a SAR and its *configuration space* $\mathcal{Q}$, which is the set of its possible configurations, the *workspace* $\mathcal{W}$ is a subset of $\mathbb{R}^3$ where the robot can move and which can contain obstacles O_i. $\mathcal{A}(q)$ denotes the subset of $\mathcal{W}$, which is occupied by the SAR when it is in configuration q. The subset of *collision-free configurations* is defined as $\mathcal{Q}_{free} := \mathcal{Q} \setminus \{q \in \mathcal{Q} : \mathcal{A}(q) \cap O_i \neq \emptyset, \forall i\}$, while the subset of the workspace which is accessible by the SAR is $\mathcal{W}_{free} := \{\mathcal{A}(q) : q \in \mathcal{Q}_{free}\}$.

B. Robot Workspace

A typical SAR includes a complex motion control system with several Degrees Of Freedom (DOF). Fig.1 shows a SAR with six different axis, called a 6-DOF robot. Modelling exactly the 3D-space reacheable by the SAR is almost impossible as it requires to analyze all the kinematics of the 6-DOF robot [10]. A well-known method to address this issue consists in simplifying the configuration space by over-approximating the shape of the robot and the various obstacles [1].

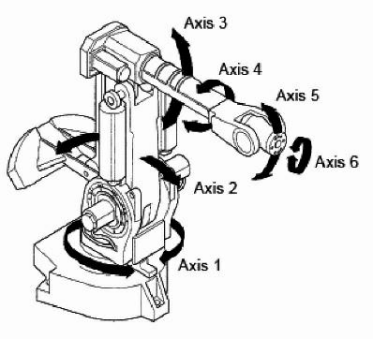

Fig. 1. Degrees of Freedom of A Single-Arm Industrial Robot

There are two distinct models for representing robot configurations, namely the *joint space configuration* which uses all the kinematics of 6-DOF robot and the *Cartesian space configuration* which represents the possible configurations of the robot in a 3D Cartesian space [1]. Typical SAR includes three distinct moves: linear motion (*moveL*), in which the robot moves from one point to another using a straightline move; circular motion (*moveC*) where a circular arc is used to move from one point to another, and non-linear joint motion (*moveJ*) where all axis of the robot move in the joint space to reach a point. In this paper, we restrict ourselves to linear motions in the Cartesian space only, as handling circular and non-linear joint motions requires to develop a complete model in the robot joint space. Developing such a model would be interesting but we consider it as outside of the scope of this paper.

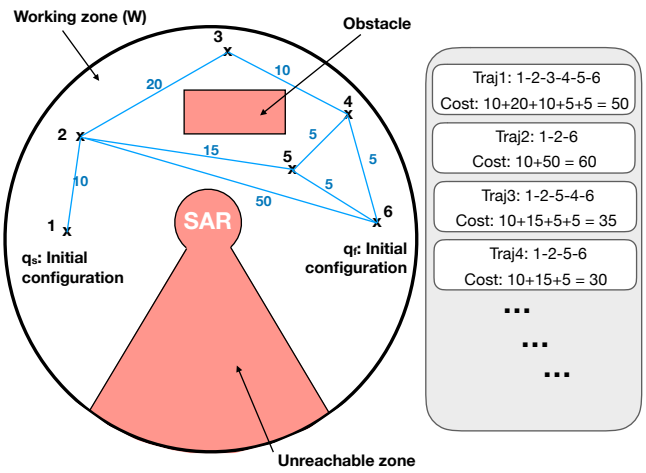

Fig. 2. 2D-Projection of the Robot Workspace

```
MODULE trajectories_test
  !robtarget start here
  VAR robtarget p10 := [[-300,-100,300],...
  VAR robtarget p20 := [[-200,438,300],...
  VAR robtarget p30 := [[50,-386,200],...
  VAR robtarget p40 := [[-75,-410,276],...
  VAR robtarget p50 := [[186,151,150],...
  VAR robtarget p60 := [[244,-69,98],...

  PROC main()
    ConfL\Off;
    !Enter move-instructions here
    MoveL p10, v1000, fine, tool0;
    MoveL p20, v500, fine, tool0;
    MoveL p60, v500, fine, tool0;
  ENDPROC
ENDMODULE
```

Fig. 3. RAPID code corresponding to Traj2.

C. Problem Formalization

The problem addressed in this paper can be formalized as such: Given a configuration space $\mathcal{Q}$, an initial (resp. final) configuration $q_s \in \mathcal{Q}$ (resp. q_f), a cost-function $c : \mathcal{Q} \times \mathcal{Q} \longrightarrow \mathbb{R}$ which associates a cost value to each possible transition, the problem aims at finding a path p that connects q_s to q_f that 1) avoids to collide any obstacle and 2) maximizes the cost of the overall path.

From the above formulation, it is important to stress the following: Moving from q_s to q_f is a continuous function which involves continuous moves of the SAR w.r.t. time. As we are interested in generating test scenarios we decide to discretize the configuration space by considering only a finite subset of configurations. For that, we considered an initial cloud of points in the robot workspace and considered only the SAR configurations related to these points of the Cartesian space.

The cost function c encompasses several elements that need to be presented. As the overall goal is to generate test scenarios which place the SAR into stressing conditions, c is associated to the complexity of SAR moves. It abstracts robot's speed and acceleration, distance between points, types of move, CPU-load of the moves, etc. In this paper, c is a given parameter which, for each pair of configurations, gives an abstract value depending of all these ingredients. The cost of the path is simply the sum of the cost of each individual transition. Note that obtaining these values requires many measurements and trials in simulation, but presenting these measurements in depth was considered to be outside the scope of this paper.

Fig.2 shows an example of the configuration space with a cloud of 6 points and the cost of four distinct paths (Traj1 to Traj4).

The programming language used to pilot the SAR is called RAPID and contains instructions to move the SAR from point to point. The language itself is very simple; Fig.3 shows an example of RAPID code, corresponding to Traj2 given in Fig.2.

From the initial cloud of points, to obtain a trajectory that avoids all the obstacles and runs through the available workspace, the method works into two steps. The first step examines all pairwise combination of points and consider intrinsic cost associated to each arc. From this step, a cost-labeled oriented graph is produced where only reachable points are indicated. The second step aims at exploring all the possible trajectories and selects one which has the maximal cost. The next section explains in depth the principle of this two-step method. B

III. Optimal Constraint-Based Generation of Continuous Trajectories

Our approach called *RobTest* aims at testing of SAR by using Constraint Programming (CP) methods and tools [11]. Two advanced CP methods, namely constraint refutation over continuous domain and constraint optimization over graphs, are used in *RobTest* to deal with the modelling of the robot's workspace and the cost maximization problem. As a result, *RobTest* includes two components named *Continuous Constraint Based Generation (CCBG)* and *Trajectories Constraint Optimization (TCO)* which implements these methods. The CCBG component creates an oriented graph by determining which pair of points is reachable and which one is not while the TCO component looks for cost-optimal loop-free trajectories in this oriented graph. *RobTest* is a fully automated approach for supporting software engineers in their test scenarios writing tasks. A general overview of *RobTest* is given in Fig.4. While the current existing process consists in writing RAPID test code, the newly introduced approach *RobTest* supports automated generation of test scenarios where trajectories with optimal cost are generated. The parameters that need to be set up for the test engineer are limited to a minimal cost value C_{max} to avoid searching for useless trajectories, a maximal number of iterations over each node N_p and a time contract for the trajectory generation N_l. The generated scenarios can be deployed and executed using a complete test execution chain, on simulators, also called virtual controllers, or on real SAR as shown in Fig.4.

In the rest of the section, we present CCBG which exploits a continuous-domains constraint solver and TCO which extracts a trajectory with maximum cost. The section is ended by a brief presentation of some generated loop-free trajectories and how they are exploited to generate SAR test cases.

A. Continuous Constraint Based Generation (CCBG)

CCBG addresses a reachability problem for each pair of points given in the initial cloud. It also calculates the cost c associated to each pair of points. The output of CCBG is a cost-labelled oriented graph where there is an arc between two points if their corresponding cost is finite (as opposed to infinite cost for non-reachable pair of points).

The implementation of CCBG is based on a continuous-domain constraint solver, which is a well-studied subtopic of CP [11]. In such a solver, any variable takes its values in a continuous domain with floating-point boundaries. As the geometry of the distinct zones and moves of the robot is composed of non-linear equations and inequations, we selected continuous domains constraint solving to address the problem. For the implementation, we selected *RealPaver* [12] among

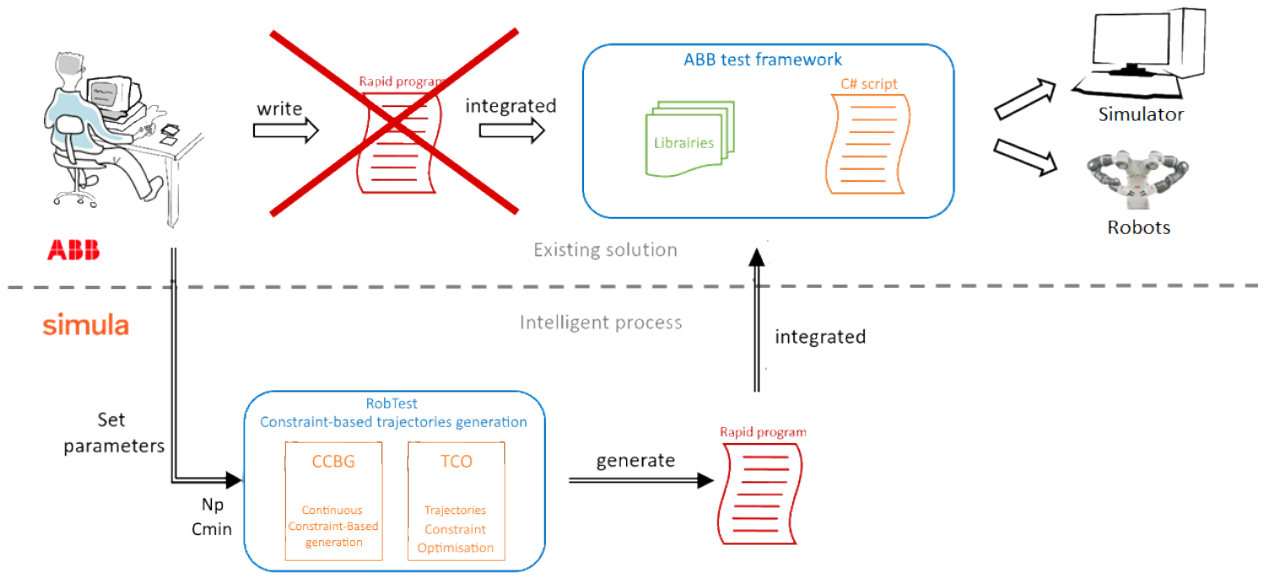

Fig. 4. A general overview of the *RobTest* approach

other available solvers such as *Ibex* [13] or *Numerica* [14] [15]. We chose this solver because it can provide both under- and over- approximation of the solution set and also because of its availability and usage simplicity.

Based on the model of the workspace $\mathcal{W}$, a pair of points can be shown reachable by the robot if a corresponding constraint system is shown unsatisfiable. Constraint solving over continuous domains works by successively pruning the domains of each variable by using each constraint as a filter. By decomposing constraints into individual projectors and iterating the application of these projectors over the domains by using a specific filtering consistency, the constraint solver is able to reach a fix point where no more pruning can be performed. The termination of the algorithm is guaranteed by the usage of abstract numerical values which are computed over floating-point values. As the number of floating-point values is finite (if standard fixed-size representation is used over 32, 64 or 128 bits), and as only domain shrinking is possible, the algorithm necessarily terminates on fixpoint.

We modeled the reachability problem into a 3D-cartesian space. Equations and inequations are used to over-approximate $\mathcal{W}$. With this model, it becomes possible to check the non-existence of intersection between the straight line passing by the pair of points and any unreachable zone corresponding to the various obstacles and robot. If the solver proves the absence of solutions (i.e., by showing unsatifiability), then it means that the pair of point is reachable, as there is no intersection between the robot's movement and any of the obstacles. Once the absence of solution is proved, the cost between the pair of nodes is calculated and added as a label on the corresponding arc of the graph. If the solver returns an under-approximation of the solution set (with a guarantee of the existence of at least one solution [12]), it means that there exists an intersection between the line and a forbidden zone, and thus the points are unreachable.

Extending this computation to other possible robot's movements is possible but requires more in-depth examination. In fact, extending the model to *moveC* (circle-arc between three points) requires to solve a variation problem when only two points are initially given. We need to generalize the previous principle for all possible triples where only two points are known. This is an interesting problem but we considered it as being outside of the scope of this paper.

In order to obtain the cost-labelled oriented graph, CCBG executes several requests over the RealPaver model of the workspace $\mathcal{W}$. The first request ensures that a straightline passing by each pair of points is included into the sphere which delimits $\mathcal{W}$ (in blue on Fig.5). The second request ensures that the unreachable zone (in red in Fig.5) is not crossed by this straightline. If some obstacles are present into $\mathcal{W}$, each obstacle will be checked with a specific query. It is worth noticing that the time-complexity of CCBG depends only on the number of points N of the initial cloud and the number of obstacles. The quadratic time-complexity is given by the following formula:

$$CCBG_{time} = \frac{N * (N-1)}{2}(RP_{sphere} + RP_{unreach} + RP_{obst}) \quad (1)$$

We illustrate CCBG on a simple example given in Fig.5, where $N = 5$ and there is only one obstacle. The generated graph contains labeled arcs only for points that can be joint using straightlines.

B. Trajectories Constraint Optimization (TCO)

TCO is the component responsible of trajectories generation. This component exploits CP over finite domain and optimization over graphs [11]. As inputs, it takes the oriented labeled graph genereated by CCBG and a user-defined value which corresponds to a minimal cost associated to the desired

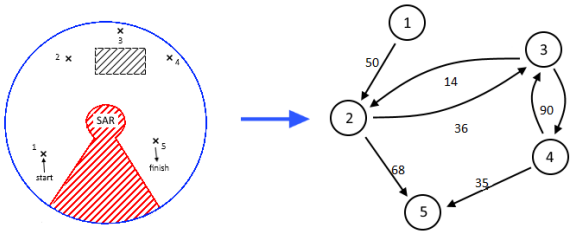

Fig. 5. Example of labeled graph generated by CCBG

trajectory. This value allows the component to prune the search space with all trajectories having a cost inferior to this value.

TCO defines a CP model with one finite domain variable X_i per node i in the graph. The domain of X_i is composed of the possible followers of i in the graph, augmented with a special value to indicate termination. For instance, the example of Fig.5 can be encoded with the following finite domain variables (where the value 0 indicates no follower): $X_1 \in \{0,2\}, X_2 \in \{0,3,5\}, X_3 \in \{0,2,4\}, X_4 \in \{0,3,5\}, X_5 \in \{0\}$. With this encoding, a loop-free trajectory in the graph is for example: $X_1 = 2, X_2 = 3, X_3 = 4, X_4 = 5, X_5 = 0$ and all the possible trajectories can be explored by using a constraint-based search procedure. A first approach to search for an optimal trajectory can be implemented by using a classical backtracking search exploring in a depth-first mode the search tree of all trajectories. By pruning the trajectories which have a lower cost than the current explored trajectory, we can optimize the backtracking process and save some effort. We have called this approach *revised DFS* and it is consdered in our experimental evaluation. However, the revised DFS approach can be further refined by using the constraints in a more active way. Observing that a trajectory must be connex in the graph, not all values of a given variable have to be explored. By modifying the encoding to take into account not only the followers but also the predecssors of nodes in the graph, it is possible to encode with constraints the trajectory connexity issue.

Our implementation of TCO is based on SICStus Prolog clpfd [16]. The CP model that was designed embedds domain constraints over finite domains variable as shown above, but also global constraints such as ELEMENT and TABLE [17]. These global constraints have powerful filtering algorithms which allows us to prune the search tree in much more efficient way than the revised DFS approach. Note also that the search procedure is tuned to generate proved optimal trajectories but, as it is important to keep full control over the timing aspects, a time-contract procedure is considered. More precisely, the search is allocated a contract of time, e.g., 5,10,20 seconds, and the search is interrupted after the deadline has passed. As a result, TCO can generate near-optimal trajectories in the given contract of time. This approach is interesting for test scenarios generation as it allows us to control the time needed for the generation by compromising the quality of the result. Of course, this approach is acceptable if and only if the near-optimal trajectories which are generated are close to the optimal value. Evaluating this aspect is part of our experimental evaluation.

Once the trajectories are generated by TCO, we automatically generate RAPID code corresponding to these trajectories as shown in Fig.3 and Fig.4. The RAPID code can then be used to run a complete test scenario on either a virtual controller or a real SAR.

C. From Trajectories to Test Case Execution

Deploying the RAPID program on a virtual controller or a real SAR as a test scenario requires some adaptation that are considered in *RobTest*.

Firstly, it is important to describe the test objective of these generated scenarios. Each time a new version of the control motion software of the SAR is produced, it must be thoroughly tested in comparison with a previous version. This non-regression test aims at revealing discrepency between an idealized trajectory used as input for the SAR and materialized by a RAPID program, and the real trajectory performed by the robot. Discrepency can originate from software bugs but also hardware issue such as mechanical vibrations or mechatronic issues. Note also that physical devices can impact the trajectory of the SAR and observing these discrepencies is crucial to improve the overall quality of the software controller.

The acceptance level is defined by a user-controlled threeshold in between the idealized trajectory and the observed trajectory of the SAR. This threeshold makes the difference between a failed test scenario and a successed test. The threeshold is not only defined in terms of robot positions but also in terms of time to reach specific points. During the execution of a test scenario, the SAR samples different parameters such as timing, position, status, etc. every 10ms. The sampling is performed on real positions of the robot's arm. As a result, the idealized trajectory as well as the real trajectory can be visualized and alerts can be issued when there are too high discrepencies or deviations between them. Fig.6 shows an example of such deviations, where the blue curve is generated by *RobTest* while the orange curve is the real trajectory of the SAR.

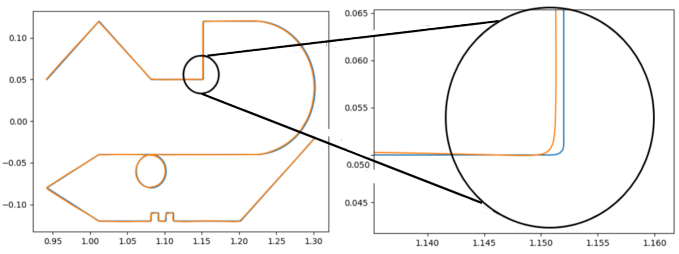

Fig. 6. Result of the execution of a test trajectory

IV. EXPERIMENTAL EVALUATION

A. Research Questions

We performed an in-depth experimental evaluation to measure the effectiveness of *RobTest* to generate loop-free,

collision-free and cost-optimal trajectories for SAR. As the ultimate goal is to embed *RobTest* in a continuous integration process and to leave it generate automatically test scenarios, we explored the tow following research questions:

RQ1 How efficient is *RobTest* in generating trajectories in a reasonable amount of time? This RQ is also very much realted to the scalability of the approach in a continuous integration process;

RQ2 Does *RobTest* benefit from the exploitation of advanced CP methods? The question is related to the usage of a constraint-based search instead of a simple DFS approach;

RQ3 How much time is needed to get acceptable near-optimal trajectories with *RobTest*;

All the experiments were performed on a regular PC, namely an Intel-core $i7$ Lenovo ThinkPad running at $2,60$GHz $T460s$ with 8GB RAM.

B. Experimental Evaluation Protocol

For our experiments, we built a generator of random points into the robot's workspace. This allowed us to repeat 10 times each experiment and take the average result, in order to avoid the bad-luck effect with random samples. We considered a number of initial points in the workspace which corresponds to the usual practice of test engineers: in between 10 to 20 points. The timing aspects were considered by keeping in mind that limited contract of time can be allocated to the trajectory generation aspects and thus we put an overall time-out of 1-hour for each individual experiment. For the sake of simplicity, we considered a cost function which is evaluated by using the pairwise distance between all the generated points. Considering a more complex cost function is of course easily feasible but it would have not bring us more interest, as the cost function is given by the test engineers under the form of a value table.

C. [RQ1]: Efficiency of RobTest

The purpose of our first experiments was to evaluate the performance of *RobTest*. Tab.I reports on the averaged CPU time (in sec) required to run the two components of *RobTest*, namely CCBG and TCO while the tool is aksed to search the global cost-optimal trajectory. Results are given under the form $val \pm dev$ which means value and standard deviation. In column **`Rho`**, the averaged density (in percentage) of the produced graph is reported. This density is an indicator of the number of unreachable pairs that were found by *RobTest*. Note that a density of 100% would mean that all pairs in the graph are reachable. From the analysis of **`Rho`** column, we observe that the produced graphs are dense (i.e., they range in between 74% and 82%). Knowing that there is a perfect positive correlation between the graph density and the number of possible paths between two points, we also observe that the search space is huge and finding an optimal trajectory is a difficult problem for the TCO component.

The results show that the time required by CCBG scales well when the number of points increases. This confirms our initial time-complexity analysis where we showed that the number of RealPaver requests depends quadratically on the number of points. According to the reported standard deviation, we also observe that the distribution of points in space does not impact the effectiveness of *RobTest*. Our approach scales up to 15 points in less than an hour computation time which is in line with the expectations of test engineers. Note however that most of time is spent in TCO, as opposed to CCBG. This does not come as a big surprise as the search process for optimal trajectory in TCO is exponential while the time-complexity of CCBG is quadratic.

D. [RQ2]: Benefice of CP methods in RobTest

The second experiment, which is also reported in Tab.I, aimed at evaluating the benefice of CP as compared to a simpler approach such as DFS. As said above, the revised DFS approach can return the optimal trajectory after having explored the search tree by backtracking. There is an optimization in this version of DFS which is based on the pruning of branches for which the cost is lower than the current cost. Tab.I.(PART II) compares the CPU time (average $\pm$ standard deviation) of both approaches, namely DFS vs TCO.

We observe that TCO is 4 to 54 times faster than DFS on the 10, 11, 12 and 13 points instances. For instances with more than 13 points, DFS exceeds the allocated time of one hour without enumerating all solutions and thus without reporting any optimal solution. Meanwhile, TCO is able to return the optimal trajectory for dense graphs of nodes up to 15. TCO needs more than one hour to return the optimal solution for dense graphs (i.e., more than 80%).

Even though both approaches (DFS and TCO) suffers from a combinatorial explosion, the constraint-based approach of TCO scales up better than DFS. Actually it scales up to 15 points which correspond the expectations of test engineers. Note that in these experiments the global optimal trajectory has been found, including their proof of optimality.

E. [RQ3]: Considering Near-Optimal Solutions

The third experiment aimed at evaluating *RobTest* when a limited contract of time is given. As the overall goal is to deploy *RobTest* in a continuous integration process, it is mandatory to fully control the time that is allocated to the generation of trajectories, even if it degrades the quality of solutions.

Fig.7 shows the averaged ratio between the near-optimal solutions and optimal solution within a cutoff of 5, 10, 15 and 20 seconds. We report the averaged ratio on the various instances (10 to 18 points).

For 10, 11 and 12 points, *RobTest* returns the optimal solution within a cutoff of 5 seconds. For the instances 13 and 14, the near-optimal solutions are returned with a ratio ranging in between 96% and 98% within a cutoff of 5 seconds. With a cutoff of 20 seconds, *RobTest* is very close to the optimal solution. (between 98% and $99,8\%$).

Let us take a closer look to the instance 15. The quasi-optimal solutions are of a ratio of, resp., 94%, 96%, $96,5\%$

INSTANCES	PART I		PART II	
#Points	**CCBG** (s)	**Rho**	**DFS** (s)	**TCO** (s)
10	2.38 ± 0.05	0.76 ± 0.04	0.48 ± 0.22	0.13 ± 0.06
11	2.91 ± 0.05	0.74 ± 0.05	3.16 ± 1.51	0.47 ± 0.49
12	3.49 ± 0.08	0.76 ± 0.06	41.83 ± 27.86	1.59 ± 1.84
13	4.13 ± 0.04	0.75 ± 0.08	597.91 ± 646.14	11.07 ± 16.61
14	4.82 ± 0.03	0.80 ± 0.03	TO	256.00 ± 403.34
15	5.56 ± 0.05	0.79 ± 0.04	TO	$876.55 \pm 1,247.29$
16	6.36 ± 0.01	0.82 ± 0.05	TO	TO

TO: a timeout of $3,600s$

TABLE I
RobTest RESULTS.

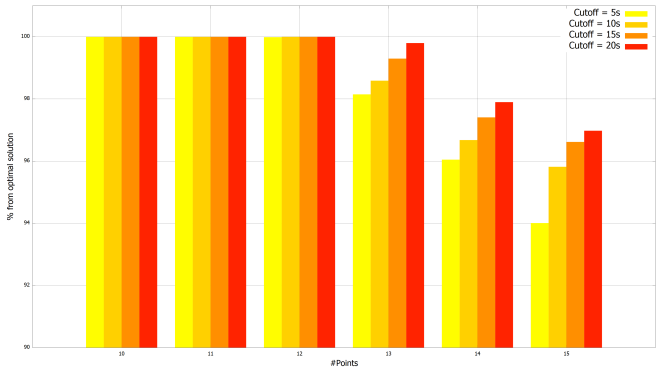

Fig. 7. Near-optimal solutions within cutoffs.

and 97% within a cutoff of, resp., 5, 10, 15, 20 seconds. That is, with only 5 seconds for such large instance the solution is of a high-quality while reaching the global optimal solution needs more than 50 minutes.

F. Discussion

Even though our experimental evaluation suffers from some internal threats-to-validity that are discussed afterwards, we observe that the efficiency of *RobTest* is demonstrated for initial cloud of points containing in between 10 to 18 points. *RobTest* would not be deployable without advanced constraint-based methods. The difference between TCO and DFS in terms of runtime is significant. At last, deploying *RobTest* in a continuous process is feasible as, according to the test engineers, the near-optimal solutions obtained in very short period of time are sufficient to guarantee an efficient testing process.

However, as said above, there is a number of internal threats-to-validity in our experiments. Firstly, we generated input points at random while the real testing process involves points which are manually generated. In order to tame the risk, we generated 10 times those points and showed the average result, plus standard deviation. However, our random generator assumes an unfiform probablity distribution over the workspace which may not correspond to reality. Secondly, in our experiment, we considered only the unreacheable zone of the SAR without introducing any obstacle in the workspace. As a consequence, we observed high percentage value for the density of the generated graphs. The introduction of many obstacles may lower the graph density and introduce a bias in the results. Further experiments are needed to evaluate the importance of graph density on the timing aspects. This is part of our perspectives for this work. Thirdly, in our experiments, we have considered a cost function computed with the pairwise distance between the randomly generated points. Even though this is acceptable as the distance is certainly a crucial element of the cost function, it is a random function in our case as the points are generated using an unfirm distribution. Using a random function for the cost of trajectory can thus be biased when compared to real trajectory which consider much more complex cost functions. Here again, we feel that more experiments are needed to fully evaluate the potential deployment of *RobTest*.

V. CONCLUSION AND PERSPECTIVES

In this paper, we introduced *RobTest* a new method for generating collision-free and loop-free cost-optimal trajectories for stress testing of industrial single-arm robots. *RobTest* exploits advanced Constraint Programming methods based on continuous domains constraint solving and constraints optimization over graphs. Our experimental evaluation of *RobTest* shows that it is efficient for dealing with intial clouds of points containing up to 18 points, a number which is considered as appropriate by test engineers. We also showed that without constraint optimization over graphs, *RobTest* is not sufficiently performant to be deployed. Its deploymemt in a continuous integration process is conditionned to fine-control of the timing aspects of the trajectory generation. By slightly compromising the quality of solution, we showed that reaching near-optimal solutions with *RobTest* is feasible in an handful of seconds. According to our knowledge, the work presented in this paper is the first complete attempt to deploy constraint-based methods in a test trajectory generation process for stress testing of industrial single-arm robot. The perspectives of this work include an increased experimental evaluation taking into account the addition of multiple obstacles in the robot's workspace. Furthermore, we plan to improve *RobTest* by considering 1) other types of robot movements such as circular motions and 2) the generation of more complex trajectories with loops.

ACKNOWLEDGMENT

This work is supported by the Research Council of Norway (RCN) through the research-based innovation center Certus,

under the SFI programme.

REFERENCES

[1] J.-C. Latombe, *Robot Motion Planning*, ch. 1. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[2] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Trajectory planning in robotics," *Mathematics in Computer Science*, vol. 6, no. 3, pp. 269–279, 2012.

[3] L. Jaulin, "Path planning using intervals and graphs," *Reliable Computing*, vol. 7, no. 1, pp. 1–15, 2001.

[4] B. Desrochers and L. Jaulin, "Computing a guaranteed approximation of the zone explored by a robot," *IEEE Transactions on Automatic Control*, vol. 62, no. 1, pp. 425–430, 2017.

[5] S. Rohou, L. Jaulin, L. Mihaylova, F. Le Bars, and S. M. Veres, "Guaranteed computation of robots trajectories," *Robotics and Autonomous Systems*, vol. 93, pp. 76–84, 2017.

[6] J. Minguez, J.-P. Laumond, and F. Lamiraux, "Motion planning and obstacle avoidance," in *Springer Handbook of Robotics*, pp. 827–852, Springer, 2008.

[7] M. Stilman, *Global Manipulation Planing in Robot Joint Space with Task Constraints*, vol. 26, pp. 576–584. IEEE Transactions on Robotics, 2010.

[8] C. Faria, F. Ferreira, W. Erlhagen, S. Monteiro, and E. a. Bicho, *Position-based kinematics for 7-DoF serial manipulators with global configuration control, joint limit and singularity avoidance*, vol. 121, pp. 317–334. Mechanism and Machine Theory, 2018.

[9] S. Pellegrinelli, A. Orlandini, N. Pedrocchi, A. Umbrico, and T. Tullio, "Motion planning and scheduling for human and robot collaboration," *CIRP Annals - Manufacturing Technology*, vol. 66, no. 1, pp. 1–4, 2017.

[10] J. García de Jalón, J. Cuadrado, A. Avello, and J.-M. Jimenez, *Kinematic and Dynamic Simulation of Rigid and Flexible Systems with Fully Cartesian Coordinates*, vol. 268, pp. 285–323. Computer-Aided Analysis of Rigid and Flexible Mechanical Systems, NATO Science Series, Springer, 1994.

[11] F. Rossi, P. Van Beek, and T. Walsh, eds., *Handbook of Constraint Programming*, vol. 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[12] L. Granvilliers and F. Benhamou, "Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques," *ACM TOMS*, vol. 32, no. 1, pp. 138–156, 2006.

[13] G. Chabert, *IBEX (Interval-Based EXplorer): A C++ library for solving nonlinear constraints over real numbers*. http://www.ibex-lib.org/, 2007.

[14] P. Van Hentenryck, M. Laurent, and D. Yves, *Numerica: A Modeling Language for Global Optimization*, vol. 228, ch. 4, pp. 57–74. USA: MIT Press, 1997.

[15] P. Van Hentenryck, "A gentle introduction to NUMERICA," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 209–235, 1998.

[16] M. Carlsson, G. Ottosson, and B. Carlson, "An open-ended finite domain constraint solver," in *Programming Languages: Implementations, Logics, and Programs, 9th International Symposium, PLILP'97, Including a Special Trach on Declarative Programming Languages in Education, Southampton, UK, September 3-5, 1997, Proceedings*, pp. 191–206, 1997.

[17] M. T. Khong, Y. Deville, P. Schaus, and C. Lecoutre, "Efficient reification of table constraints," in *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, pp. 118–122, 2017.