2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

# On Optimizing a Sequence of Robotic Tasks

Sergey Alatartsev, Vera Mersheeva, Marcus Augustine and Frank Ortmeier

***Abstract*— Production speed and energy efficiency are crucial factors for any application scenario in industrial robotics. The most important factor for this is planning of an optimized sequence of atomic subtasks. In a welding scenario, an atomic subtask could be understood as a single welding seam/spot while the sequence could be the ordering of these atomic tasks. Optimization of a task sequence is normally modeled as the Traveling Salesman Problem (TSP). This works well for simple scenarios with atomic tasks without execution freedom like spot welding. However, many types of tasks allow a certain freedom of execution. A simple example is seam welding of a closed-contour, where typically the starting-ending point is not specified by the application. This extra degree of freedom allows for much more efficient task sequencing. In this paper, we describe an extension of TSP to model a problem of finding an optimal sequence of tasks with such extra degree of freedom. We propose a new, efficient heuristic to solve such problems and show its applicability. Obtained computational results are close to the optimum on small instances and outperforms the state of the art approaches on benchmarks available in literature.**

## I. INTRODUCTION

Production efficiency and quality are key success factors for highly developed industrial countries. One major cornerstone to achieve this is to rely on production automation technology. Industrial robots are among the most flexible and powerful of these machines. Equipped with the right tool, they can be applied to almost every production task. The faster industrial robots do their work, the more income they provide.

The time required for a given application activity heavily depends on sequencing of the effective tasks/movements. By effective tasks we refer to such movements, which are defined by the application itself. For example, welding seams or contours which have to be cut. Supporting movements/tasks are in between such effective tasks. They are necessary to move the tool from one effective task to another. Supporting movements are typically not defined by the application.

The total cost/time a robot needs for completing an activity is the sum of costs of all effective movements and all supporting movements. While the first is typically defined by the application, physical process and/or tool specifics, the later is mostly depending upon smart path-planning and the sequencing of effective tasks. In this paper, we will focus on the sequencing aspect.

S. Alatartsev, M. Augustine and F. Ortmeier are affiliated with the working group Computer Systems in Engineering, Otto-von-Guericke University of Magdeburg, Germany. {sergey.alatartsev, marcus.augustine, frank.ortmeier}@ovgu.de

V. Mersheeva is affiliated with the Institute for Applied Informatics, Alpen-Adria-Universität of Klagenfurt, Austria. vera.mersheeva@aau.at

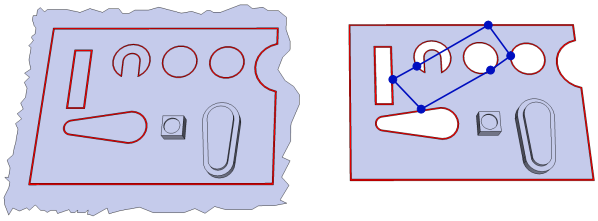

Fig. 1. A plastic detail: before being processed by a robot (on the left) and after (on the right). The optimal tour is designated with a blue line.

In contrast to existing approaches, we want to make use of the fact that most robotic tasks allow a certain degree of flexibility. Though they are not treated in such a way, i.e., specified in a mono-semantic way that describes only one variant of task execution. See extensive surveys [1], [2].

There are several industrial use cases, where extra degree of freedom exists. One such example is shown in Fig. 1. This example is an abstraction of a real-world automotive application. Here, a plastic panel is produced by a molding press out of one piece of plastic. Later, the excess of the melted plastic on the borders of the form as well as different shapes have to be cut out by the industrial robot. Cutting contours are shown as red lines. Each of the six single contours can be understood as an individual, atomic, effective task. Obviously, the description of these contours alone is not sufficient to process the detail, as a robot path is required. In practice, the path is derived from CAD data in an off-line programming environment. Starting points of the tasks are allocated manually, based on intuition (See Section II). However, from an application point of view there is often no requirement that defines where every cut has to be started/ended. Therefore, it could be any arbitrary point along the closed contour. In Fig. 1 such points have been chosen. The blue lines represent the necessary supporting movements[1]. As it is shown in the illustration, even with six contours the optimal tour could be non-trivial to construct. This paper focuses on a heuristic for computing a near optimal tour automatically.

The remainder of the paper is organized as follows. Section II briefly outlines state of the art approaches. In the Section III, problem definition and involved algorithms are introduced. Section IV provides description of the proposed approach. An evaluation is given in Section V. We conclude and provide an outlook to future work in Section VI.

[1]The movements are projected into 2D space and assumed as linear movements for simplicity. In a real application one might use Point-to-Point (PTP) movements, which are linear in axis space but not linear in Cartesian space.

978-1-4673-6357-0/13/$31.00 ©2013 IEEE

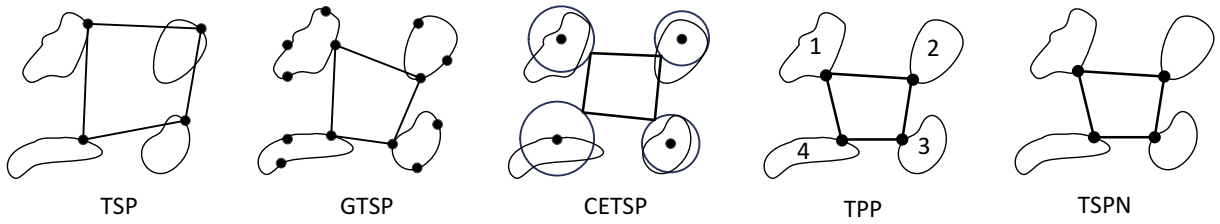


Fig. 2. The differences between sequencing problems

## II. RELATED WORK

We start with a short summary of related optimization problems. The differences between them are illustrated in Fig. 2. Further we give an overview of existing algorithms for solving such problems.

### A. Modeling problems

The problem of choosing an optimal sequence of effective tasks is normally represented as the Traveling Salesman Problem (TSP) [3]. The goal of the TSP is to find a minimal-cost circle tour through a set of points such that every point is visited once. Every task in TSP is represented as one point. It works well for the tasks with limited freedom, e.g., spot-welding applications. But it is inefficient for tasks with flexibility in execution, e.g., cutting a closed-contour segments.

There are several well-known TSP-like problems that allow a certain flexibility in task description. It is possible to represent any task with a set of points. This leads to a Generalized TSP (GTSP) [4], where the minimal-cost circle tour should be found that contains a point from every set. The representation of complex areas requires a large number of points that will cause higher computational efforts. In addition, a certain level of discretization brings an error to the final result.

Another way of modeling the sequencing problem is Close-Enough TSP (CETSP) [5], where the minimal-cost circle tour should be found such that every point is visited within a certain radius. This problem does not have the discretization error in contrast to GTSP. Nevertheless, many robotic tasks can not be efficiently approximated with circles.

One more related problem is Touring-a-sequence-of-Polygons Problem (TPP) [6]. The goal of the TPP is to find the minimal-cost circle tour that visit a *predefined sequence* of regions. Although this problem can find the optimal point positions inside the given areas, it is not able to calculate the optimal sequence.

The most general problem is TSP with Neighborhoods (TSPN) [7]. The goal of the TSPN is to find the minimal-cost circle tour through a set of regions such that every region is visited once. TSPN is the most flexible problem in comparison to those discussed above. The regions could be arbitrarily shaped. It has no discretization error and it is capable of optimizing both points allocation and their sequence.

### B. Existing algorithmic solutions

Previously, TSPN was mostly covered by researchers from the approximation domain. The reader can find comprehensive surveys in [8] and [9]. Over the time, TSP got much larger attention from the researchers and multiple, very efficient heuristics were developed. The natural idea is to adapt existing well performing methods from the TSP to its more flexible variants. For example, Karaperyan et al. [10] proposed an adaptation of the Lin-Kernighan heuristic to GTSP. Mennell [5] proposed an approach for CETSP, when the problem is divided into the TSP and the TPP [6], which already have efficient solvers. At first, areas are represented as points and TSP tour is calculated. Further allocation of the points inside the areas is optimized with TPP. A similar concept but with a different optimization order was proposed for TSPN by Elbassioni et al. [11]. At first the point allocation is optimized with a greedy algorithm, when the nearest point from the next largest area is added to the set. After that a TSP tour is calculated.

Further development of the idea of splitting TSPN into smaller problems was made by Alatartsev et al. [12]. They proposed a method, the so-called Constricting Insertion Heuristic (CIH), where TSPN is represented as a combination of TSP and TPP, like in previous work done by Mennell [5]. CIH solves TSPN by applying the Insertion Heuristic (IH) to TSP and the Rubber-band algorithm [13] to TPP. The main difference from the previous approaches (e.g., [11] and [5]) is that constricting and sequencing are not solved one after another, but rather simultaneously. We will compare it with the algorithm proposed in this paper in Section V.

### C. Sequence optimization in robotics

Nowadays, industrial robot off-line programming environments either ignore sequence optimization (e.g., RobotWorks[2]) or solve it by simple TSP approaches (e.g., a customization for DELMIA adapted for drilling applications[3] allows automatic sequencing of drilling tasks by a simple greedy algorithm to solve TSP).

There are several approaches that apply TSP for modeling a sequence optimization problem of robot's tasks [14], [15]. All such approaches *solely* rely on freedom in the sequence of the supporting tasks and none of them involves freedom of the effective tasks execution into sequence optimization.

[2]Compucraft Ltd, RobotWorks: www.compucraftltd.com

[3]DELMIA V5 Robotic Drilling Application: http://www.delfoi.com/web/products/delfoi_products/en_GB/drilling-app/

Recently, the idea of modeling the sequencing problem in robotics with TSPN was covered by Gentilini et al. [16]. The TSPN application was illustrated by a robot with a camera mounted on its end-effector. The task did not require a precise position but rather an area from which the pictures have to be taken. It was shown that searching for an exact solution required unreasonable time. Thus, a heuristic was introduced to a Mixed-Integer Non-Linear Programming solver to speed up the calculation time (we will refer to it simply as HIS, i.e., Heuristic in Solver). The method was evaluated on the test instances with up to 16 areas. Comparison of the results of our method and HIS will be presented in Section V.

In this paper we suggest to use TSPN for sequence optimization of robot's tasks. We make use of two previously applied principals: 1) split TSPN to TSP and TPP and 2) solve TSP and TPP simultaneously. We go a step further and – in contrast to related approaches – apply a tour-improvement heuristic from TSP domain to solve TSPN. We refer to this approach as Constricting 3-Opt (C3-Opt).

## III. PRELIMINARIES

This section presents a formal definition of the TSPN and basic ideas of the applied algorithms.

### A. Problem definition

We suggest to model robot sequence optimization with the TSPN that is formalized as follows:

**Input:** a set of $n$ areas $A = \{A_1, ..., A_n\}$
**Goal:** find an optimal sequence and locations of the points $p_1, ..., p_n$ in the corresponding areas $A_1, ..., A_n$
**Output:** a minimal-cost cyclic tour $T = (p_1, ..., p_{n+1})$ such that it visits $A_i$ in the point $p_i$ and $p_1 = p_{n+1}$.

TSPN could be understood as a fusion of TSP and TPP. Its objective combines the goals of TSP and TPP. TSPN input is the most general among both – a set of areas. The output is inherited from TPP.

### B. Involved algorithms

*1) 3-Opt algorithm:* The $K$-exchange algorithm was developed by Lin [17]. The basic idea is to sequentially select $K$ edges from the tour and reconnect them in all possible ways. If the reconnection causes a decrease of the tour cost, the algorithm starts from the beginning; otherwise, the next $K$ edges are selected. The algorithm stops when there are no more exchanges that could improve the tour.

Evaluation [18] shows that if $K$ is greater than three, the effectiveness of the approach decreases. Therefore, normally the case $K$=3 is used. This method is referred to as 3-Opt [19]. Later 3-Opt received a lot of attention and was successfully applied in different areas, e.g., the UAV routing problem [20] or the Sequentially Ordered Problem [21].

There are seven possible ways of reconnecting three edges in 3-Opt. It was shown in [22] that only two ways of reconnecting are required to cover all possible combinations,

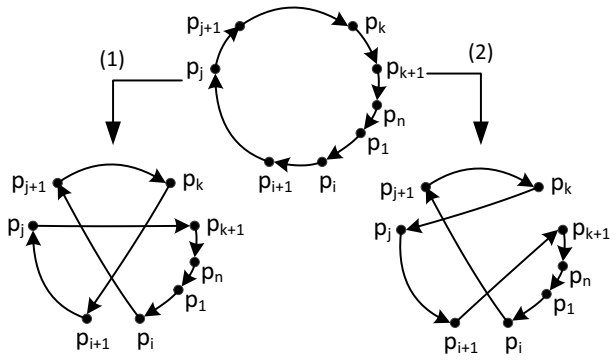


Fig. 3. Two possible replacements of edges $(p_i, p_{i+1})$, $(p_j, p_{j+1})$ and $(p_k, p_{k+1})$ in 3-Opt

see Fig. 3. The decrease of the cost in 3-Opt is calculated as the difference between the costs of newly added and deleted edges.

*2) Rubber-band algorithm:* The Rubber-band algorithm (RBA) was proposed by Pan et al. [13] to solve TPP. It takes a sequence of areas $A = (A_1, ..., A_n)$ and returns a tour $T = (p_1, ..., p_n)$ such that every point $p_i$ from the tour belongs to the area $A_i$ and the tour length is minimal. RBA's basic idea is to walk through the sequence $A$ and for every $A_i$ find a point $p_i \in A_i$ so that $d(p_{i-1}, p_i) + d(p_i, p_{i+1})$ is minimized. RBA optimizes positions of the points one by one in every area. One iteration is finished when position of every point $p_i$ is optimized. RBA stops when difference of the tour lengths between current and previous iteration is less than accuracy $\varepsilon$.

Let $PointConstrict(p_{i-1}, A_i, p_{i+1}, \mu)$ denote a function that returns a position of the point $p_i \in A_i$ optimized towards points $p_{i-1}$, $p_{i+1}$ with accuracy $\mu$. Points $p_{i-1}$, $p_{i+1}$ belong to the neighboring areas $A_{i-1}$ and $A_{i+1}$ respectively.

Any geometric or optimization approach could be applied for searching the new $p_i$, e.g., the Golden search method was applied in CIH [12]. In this paper, a simple one-dimensional Bisection method is applied. The optimal location of $p_i$ with respect to its neighboring points is illustrated in Fig. 4.

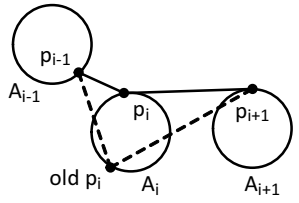


Fig. 4. Constricting point $p_i$ to its neighboring points $p_{i-1}$ and $p_{i+1}$. Edges before constriction are marked by dash lines. Edges after the constriction are marked by solid lines.

## IV. ALGORITHM

In this section we will describe the proposed tour-improvement heuristic Constricting 3-Opt (C3-Opt) that can solve TSPN efficiently. It is an extension of the classic 3-Opt that is successfully applied in the TSP domain. The main

difference of C3-Opt to 3-Opt is that it also involves a TPP solver (in our case RBA), i.e., it is able to answer where the points should be located inside the areas. In the original 3-Opt these points are fixed.

In general tour-improvement heuristics are slower than tour-construction methods. The adaptation to TSPN also brings extra expenses of computational time. Therefore, we provide a description of several techniques that allow speeding up calculation time and minimizing risks of getting into local optima fast.

### A. Constricting 3-Opt general idea

General structure of the C3-Opt is presented in Algorithm 1. The input of the C3-Opt is a set of areas $A$, an initial tour $T'$ and desired accuracies $\varepsilon$, $\mu$. Three main loops are started in lines 4–6. Indices $i$, $j$, $k$ are used to select the edges that have to be exchanged. C3-Opt uses the same two ways to reconnect the edges as in 3-Opt. Therefore, the overall logic could be split into two segments denoting each reconnection variant: the first is in lines 7–13 and the second is in lines 14–20. A candidate tour $candT$ is constructed with the function $NewTour1$ by applying the first variant of reconnections to the original tour. If the candidate tour was successfully constructed, i.e., $candT$ is not empty (line 8), and it is shorter than the current tour $T$ (line 9), then the tour is optimized by the RBA algorithm. RBA does not change the sequence but only optimizes locations of the points in the areas. The optimized tour becomes a current tour $T$ (line 10). The length of the reduced tour is saved in $Lgth_T$. After that C3-Opt continues the optimization process with the first set of edges as the original 3-Opt. Otherwise, it can miss some possible improvements occurred after reconnection. Therefore, main loops are restarted in lines 11 and 18. The second variant of reconnection (lines 14–20) has a similar logic. In contrast to the first variant, it applies another tour construction function, $NewTour2$, due to the alternative way of reconnection of the tour (see Fig. 3). In the next subsection we will show how the candidate tour is constructed. The algorithm stops when there are no more possible reconnections. The output is the optimized tour $T$.

### B. Construction of the candidate tour

3-Opt compares sums of the costs of the new and deleted edges to check whether the exchange improved the solution or not. It is straightforward, as the cost of all other edges is constant. However, in TSPN, points could change their locations within the areas. Therefore, a new exchange could bring the improvement not only by changing the sequence, but also by optimizing locations of the points inside the areas. A trivial solution could be to apply RBA algorithm on the newly constructed tour. However, in practice it leads fast to the local minimum (see Section V). We propose another method: after adding a new point $T_{last}$ to the tour, the location for the previously added point $T_{last-1}$ should be recalculated, i.e., function $PointConstrict(T_{last-2}, A_{last-1}, T_{last}, \mu)$ should be applied. This method of the tour constricting does not return the optimum point positions within the areas, but only slightly improves the tour. In practice, this improvement prevents the method to get to a local optimum fast.

**Algorithm 1:** Constricting 3-Opt

**Input**: Set of areas $A = \{A_1, ..., A_n\}$, initial tour $T' = (p'_1, ..., p'_n)$ (so that $p'_i \in A_i$), accuracies $\varepsilon$, $\mu$
**Output**: Tour $T = (p_1, ..., p_n)$

```
1  T ← T';
2  Lgth_T ← Length(T);
3  candT ← null;
4  for i = 1; i ≤ n − 2; i++ do
5      for j = i + 1; j ≤ n − 1; j++ do
6          for k = j + 1; k ≤ n; k++ do
7              candT←NewTour1(T, Lgth_T, μ, i, j, k, n);
8              if candT ≠ null then
9                  if Length(candT) < Lgth_T then
10                     T ← RBA(candT, ε, μ);
11                     GoTo(line 2);
12                 end
13             end
14             candT←NewTour2(T, Lgth_T, μ, i, j, k, n);
15             if candT ≠ null then
16                 if Length(candT) < Lgth_T then
17                     T ← RBA(candT, ε, μ);
18                     GoTo(line 2);
19                 end
20             end
21         end
22     end
23 end
24 return T;
```

The candidate tour for the first reconnection case (see Fig. 3) is constructed by the Algorithm 2. It copies the points from the current tour $T$ one by one to the new tour $candT$.

Points are added with the method $AddAndPointConstrict$, that basically adds point $T_h$ to the end of the tour $candT$ and then constricts the previously added point $candT_{last-1}$ by using $PointConstrict(candT_{last-2}, A_{last-1}, candT_{last}, \mu)$. When all the points are copied, the algorithm returns the tour.

Note that at some point in time the length of the candidate tour could exceed the length of the current tour. In that case it makes no sense to perform further copying-constricting actions and, therefore, the algorithm returns a null tour. This check could be performed multiple times at any stage of creating a candidate tour. However, in case of earlier or multiple appliances, the expenses of the check could exceed the benefit from it. We propose to locate it after the third loop in the lines 11–14 of the Algorithm 2.

The algorithm $NewTour2$ for the second variant of reconnection is constructed in the same way as $NewTour1$; the only difference is that points from $p_{i+1}$ to $p_j$ should be copied in reverse order.

### C. Constricting algorithm

The goal of the constricting algorithm was already covered in Section III-B.2. Further, we will give hints on our specific realization. The objective is to find a point on the border

**Algorithm 2:** NewTour1

**Input**: $T$, $Lgth_T$, $\mu$, $i$, $j$, $k$, $n$
**Output**: Tour $candT$

```
candT ← null;
for h = 1; h ≤ i; h++ do
    candT ← AddAndPointConstrict(candT, T_h, μ);
end
for h = j + 1; h ≤ k; h++ do
    candT ← AddAndPointConstrict(candT, T_h, μ);
end
for h = i + 1; h ≤ j; h++ do
    candT ← AddAndPointConstrict(candT, T_h, μ);
end
if Length(candT) > Lgth_T then
    candT ← null;
    return candT;
end
for h = k + 1; h ≤ n; h++ do
    candT ← AddAndPointConstrict(candT, T_h, μ);
end
PointConstrict(candT_{last−1}, A_last, candT_1, μ);
return candT;
```

of the area that is the nearest to its neighboring points. In the used 2D instances, the border is one-dimensional. The previous work [12] applied the Golden Cut method [23] to search for the local minimum on the interval from 0 to 360 degrees. For every selected degree a point on the area border is calculated. However, we are using the Bisection method because in our evaluation it turned out to be more efficient.

## V. EVALUATION

In this section we compare C3-Opt with state of the art approaches on two test instance sets derived from literature. The first evaluation is conducted on the small instances with known optimum. Further, we show the efficiency of C3-Opt on large tests with differently "stretched" ellipses. Due to the limited space, only the most important results are presented here. The full evaluation is available on-line [24].

We compare C3-Opt with state of the art approaches: HIS, Constricting Insertion Heuristic (CIH)[4] [12] and CIH (3-$Impr.$). CIH (3-$Impr.$) is a variant of CIH where 3-Opt and RBA are used afterwards to improve the solution.

C3-Opt is a tour-improvement heuristic, and its efficiency depends on the input tour. We evaluate C3-Opt with three different input tours generated by the Nearest-Neighbor algorithm (NN) (i.e., choose the nearest point next), randomly (i.e., as the ellipses are listed in the test file) and CIH. Starting points for the listed tour-construction heuristics were set to the centers of the ellipses.

All the methods (except HIS and solver for optimal values) were ran on the following hardware: Intel Core 2 Quad CPU, 2.83GHz with 8GB of RAM, running Microsoft Windows Vista. HIS and solver for optimal values were ran by Gentilini et al.[16] using Intel Xeon, 3.33GHz CPU with 12GB of RAM, running Fedora. Since the provided computational time was obtained in different conditions, it cannot be compared directly.

[4]Note that the obtained results of CIH are slightly better than presented previously in [12] due to the application of Bisection search instead of Golden search in the $PointConstrict$ function.

### A. Comparison with optimum

In this subsection, we compare C3-Opt with the existing algorithms on the test instances with known optimum. The test instances and optimal values were provided by Gentilini et al. [16]. The test data is available on-line[5]. Instance name "$tspn2DE7\_N$" denotes that the 2D test consists of 7 ellipses. "$N$" could be either "1" or "2" and denotes the size of the ellipses. Ellipses are larger in tests with "1" than in tests with "2". The C3-Opt was executed with the following parameters: $\varepsilon$=20 and $\mu$=20.

The results of the evaluation are presented in Table I. Surprisingly, the use of the greedy NN heuristic to calculate the input tour for C3-Opt led to worse results. The average and the maximum errors are 0.19% and 2.39% respectively. The average computational time is $38.21ms$. It could be explained by the fact that C3-Opt is a local search algorithm, therefore, it has no efficient techniques to escape from the local minimum established by NN.

Almost the same solution quality was shown by Rand→C3-Opt and CIH→C3-Opt that have an average error of 0.003% and 0.001% respectively (for 24 used benchmarks). The time with CIH input tour is significantly shorter than with Random tour ($28.73ms$ versus $62.64ms$). Both of these variants outperformed related algorithms in the quality of the solution.

The C3-Opt depends on the two precision parameters $\varepsilon$ and $\mu$. We used instances from Table I to evaluate their influence on algorithm performance. We selected the following values: 0.1, 1, 10, 20 and 30 and assign them to $\varepsilon$ and $\mu$ in all possible combinations. Final results were optimized by RBA. Obviously, the smaller precision parameters are, the more computation time is needed. The average time ranged from $44.94ms$ to $167.25ms$. Surprisingly, the quality of the solution weakly depends on the precision parameters. The average error differed from 0.003% ($\varepsilon$=20, $\mu$=20) to 0.05% ($\varepsilon$=10, 20 or 30, $\mu$=0.1). The evaluation shows that C3-Opt is very stable on small instances and even in the worst case the average error is not more than 0.05%.

### B. Evaluation of C3-Opt on tests with "stretched" ellipses

In this section we present an evaluation on the large test instances with differently stretched ellipses. These test instances were previously presented in [12] and they are available on-line [24]. Instance name "20_1_5" denotes the test with 20 ellipses, where every ellipse has a ratio between its radii along the axis from 1 to 5. In other words, instance "NN_1_1" consists only of circles (i.e., CETSP case). The larger the ratio is, the more stretched the ellipses are. The C3-Opt was executed with the following parameters: $\varepsilon$=20 and $\mu$=10.

[5]TSPN Instances: http://wpweb2.tepper.cmu.edu/fmargot/ampl.html

TABLE I

EVALUATION ON SMALL TEST INSTANCES WITH KNOWN OPTIMAL VALUE

| Instance | Optimal value | HIS error (%) | HIS time ($ms$) | CIH error (%) | CIH time ($ms$) | CIH (3-$Impr.$) error (%) | CIH (3-$Impr.$) time ($ms$) | NN→C3-Opt error (%) | NN→C3-Opt time ($ms$) | Rand→C3-Opt error (%) | Rand→C3-Opt time ($ms$) | CIH→C3-Opt error (%) | CIH→C3-Opt time ($ms$) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tspn2DE5_1 | 191.255 | 0.00 | 140 | 0.00 | 0.87 | 0.00 | 1.23 | **1.46** | 0.69 | 0.00 | 1.19 | 0.00 | 1.38 |
| tspn2DE5_2 | 219.307 | 0.00 | 130 | 0.00 | 0.57 | 0.00 | 0.67 | 0.00 | 0.56 | 0.00 | 1.29 | 0.00 | 0.84 |
| tspn2DE6_1 | 202.995 | 0.00 | 240 | 0.00 | 0.93 | 0.00 | 1.08 | 0.00 | 1.09 | 0.00 | 1.83 | 0.00 | 1.47 |
| tspn2DE6_2 | 248.860 | 0.00 | 180 | 0.00 | 0.82 | 0.00 | 0.98 | 0.00 | 0.69 | 0.00 | 1.62 | 0.00 | 1.32 |
| tspn2DE7_1 | 201.492 | 0.00 | 300 | **0.02** | 3.46 | **0.02** | 3.94 | 0.00 | 2.95 | 0.00 | 2.58 | **0.02** | 4.50 |
| tspn2DE7_2 | 239.788 | 0.00 | 250 | 0.00 | 1.78 | 0.00 | 2.04 | 0.00 | 2.27 | 0.00 | 5.86 | 0.00 | 2.87 |
| tspn2DE8_1 | 190.243 | 0.00 | 370 | 0.00 | 0.42 | 0.00 | 0.74 | 0.00 | 2.92 | 0.00 | 7.90 | 0.00 | 2.67 |
| tspn2DE8_2 | 229.150 | **0.01** | 400 | 0.00 | 3.49 | 0.00 | 4.11 | 0.00 | 3.43 | 0.00 | 4.69 | 0.00 | 5.36 |
| tspn2DE9_1 | 259.290 | 0.00 | 400 | 0.00 | 5.78 | 0.00 | 6.81 | 0.00 | 8.00 | 0.00 | 11.47 | 0.00 | 8.79 |
| tspn2DE9_2 | 262.815 | 0.00 | 410 | 0.00 | 4.58 | 0.00 | 5.30 | 0.00 | 6.12 | **0.01** | 13.62 | 0.00 | 7.78 |
| tspn2DE10_1 | 225.126 | 0.00 | 410 | 0.00 | 5.84 | 0.00 | 6.83 | 0.00 | 9.01 | 0.00 | 10.89 | 0.00 | 11.31 |
| tspn2DE10_2 | 273.192 | **0.21** | 350 | 0.00 | 5.08 | 0.00 | 6.18 | 0.00 | 16.66 | 0.00 | 17.48 | 0.00 | 10.97 |
| tspn2DE11_1 | 247.886 | **0.75** | 630 | 0.00 | 8.02 | 0.00 | 9.82 | **0.69** | 18.79 | 0.00 | 24.94 | 0.00 | 16.12 |
| tspn2DE11_2 | 258.003 | 0.00 | 390 | 0.00 | 7.37 | 0.00 | 9.04 | 0.00 | 25.65 | 0.00 | 24.08 | 0.00 | 15.46 |
| tspn2DE12_1 | 265.858 | 0.00 | 550 | 0.00 | 9.54 | 0.00 | 11.52 | 0.00 | 25.40 | 0.00 | 63.28 | 0.00 | 21.14 |
| tspn2DE12_2 | 312.493 | **0.50** | 860 | 0.00 | 11.89 | 0.00 | 13.75 | 0.00 | 66.01 | 0.00 | 72.10 | 0.00 | 23.96 |
| tspn2DE13_1 | 278.876 | 0.00 | 1150 | 0.00 | 15.24 | 0.00 | 18.49 | 0.00 | 27.05 | 0.00 | 66.96 | 0.00 | 32.58 |
| tspn2DE13_2 | 324.271 | **0.20** | 490 | 0.00 | 15.33 | 0.00 | 18.07 | 0.00 | 60.90 | 0.00 | 60.43 | 0.00 | 34.01 |
| tspn2DE14_1 | 310.794 | 0.00 | 950 | 0.00 | 22.82 | 0.00 | 26.75 | 0.00 | 85.96 | 0.00 | 99.12 | 0.00 | 45.93 |
| tspn2DE14_2 | 270.638 | **0.56** | 690 | 0.00 | 18.68 | 0.00 | 21.99 | 0.00 | 111.71 | **0.07** | 212.62 | 0.00 | 43.78 |
| tspn2DE15_1 | 289.716 | **0.22** | 1080 | 0.00 | 28.28 | 0.00 | 34.44 | 0.00 | 38.49 | 0.00 | 196.19 | 0.00 | 60.44 |
| tspn2DE15_2 | 293.357 | **0.01** | 1200 | **1.36** | 28.06 | **1.36** | 32.99 | **0.01** | 64.65 | 0.00 | 91.77 | 0.00 | 78.50 |
| tspn2DE16_1 | 369.945 | **1.09** | 2840 | **6.26** | 26.92 | **5.44** | 36.35 | **2.39** | 192.27 | 0.00 | 172.19 | 0.00 | 152.94 |
| tspn2DE16_2 | 295.130 | 0.00 | 1200 | **0.01** | 61.58 | **0.01** | 71.31 | 0.00 | 145.68 | 0.00 | 339.39 | 0.00 | 105.34 |
| Average: | | 0.148 | 650.42 | 0.319 | 11.97 | 0.285 | 14.35 | 0.190 | 38.21 | 0.003 | 62.64 | 0.001 | 28.73 |
| Maximum: | | 1.09 | 2840.00 | 6.26 | 61.58 | 5.44 | 71.31 | 2.39 | 192.27 | 0.07 | 339.39 | 0.02 | 152.94 |

TABLE II

EVALUATION ON TEST INSTANCES WITH DIFFERENTLY "STRETCHED" ELLIPSES

| Instance | Best known value | Old best known value | Old best error (%) | CIH error (%) | CIH time ($s$) | CIH (3-$Impr.$) error (%) | CIH (3-$Impr.$) time ($s$) | NN→C3-Opt error (%) | NN→C3-Opt time ($s$) | Rand→C3-Opt error (%) | Rand→C3-Opt time ($s$) | CIH→C3-Opt error (%) | CIH→C3-Opt time ($s$) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20_1_1 | 318.904 | 320.720 | **0.56** | **2.39** | 0.09 | **1.82** | 0.13 | **0.62** | 0.22 | 0.00 | 0.74 | 0.00 | 0.37 |
| 20_1_5 | 312.915 | 313.497 | **0.18** | **3.30** | 0.08 | **3.30** | 0.09 | 0.00 | 0.43 | 0.00 | 1.44 | 0.00 | 0.39 |
| 20_1_10 | 252.350 | 276.793 | **9.68** | 0.00 | 0.14 | **0.00** | 0.14 | **1.40** | 1.29 | **1.10** | 0.70 | 0.00 | 0.22 |
| 30_1_1 | 383.578 | 383.578 | 0.00 | **1.44** | 0.22 | 1.37 | 0.27 | **0.77** | 1.96 | 0.00 | 3.94 | 0.00 | 2.13 |
| 30_1_5 | 316.854 | 316.922 | **0.02** | 0.00 | 0.31 | **0.00** | 0.36 | **1.08** | 6.90 | **0.11** | 12.15 | 0.00 | 0.96 |
| 30_1_10 | 306.338 | 321.188 | **4.84** | 0.00 | 0.44 | **0.00** | 0.49 | **0.12** | 3.57 | **1.50** | 6.08 | 0.00 | 1.05 |
| 40_1_1 | 416.556 | 421.339 | **1.14** | **3.58** | 0.42 | **2.29** | 0.80 | 0.00 | 28.93 | 0.00 | 51.89 | **0.46** | 8.33 |
| 40_1_5 | 366.637 | 368.802 | **0.59** | **0.53** | 0.75 | **0.53** | 0.91 | **3.77** | 12.42 | 0.00 | 45.46 | **0.53** | 2.94 |
| 40_1_10 | 311.714 | 312.353 | **0.20** | 0.00 | 0.91 | **0.00** | 1.08 | **4.55** | 38.34 | **0.31** | 52.75 | 0.00 | 3.11 |
| 50_1_1 | 438.215 | 438.182 | 0.00 | **3.10** | 0.82 | **1.51** | 2.39 | **0.01** | 54.17 | 0.00 | 158.94 | **0.03** | 64.49 |
| 50_1_5 | 435.158 | 457.114 | **5.04** | **6.97** | 1.34 | **6.32** | 2.02 | **0.23** | 103.14 | **0.65** | 247.36 | 0.00 | 51.63 |
| 50_1_10 | 391.303 | 397.472 | **1.57** | **2.44** | 1.66 | **1.70** | 2.82 | 0.00 | 135.43 | **0.85** | 208.36 | **0.26** | 45.37 |
| 60_1_1 | 559.042 | 563.603 | **0.81** | **8.87** | 1.23 | **6.48** | 7.00 | **0.41** | 154.66 | 0.00 | 265.01 | **1.00** | 143.74 |
| 60_1_5 | 550.121 | 563.438 | **2.42** | **2.93** | 1.69 | **2.71** | 4.23 | **2.27** | 151.88 | 0.00 | 473.45 | **0.41** | 101.05 |
| 60_1_10 | 482.289 | 499.973 | **3.66** | **7.85** | 1.86 | **6.73** | 3.39 | **2.20** | 179.61 | 0.00 | 467.24 | **0.21** | 126.04 |
| 70_1_1 | 599.819 | 622.098 | **3.71** | **5.74** | 2.19 | **4.95** | 9.07 | **4.82** | 419.90 | **0.20** | 544.83 | 0.00 | 270.63 |
| 70_1_5 | 564.303 | 587.004 | **4.02** | **7.60** | 2.59 | **3.92** | 13.27 | **0.52** | 717.22 | 0.00 | 959.91 | **0.78** | 200.96 |
| 70_1_10 | 447.452 | 509.905 | **13.95** | **9.28** | 3.30 | **9.03** | 11.79 | **5.17** | 634.06 | 0.00 | 906.24 | **2.08** | 217.23 |
| Average | | | 2.91 | 3.67 | 1.11 | 2.93 | 3.35 | 1.55 | 146.90 | 0.26 | 244.81 | 0.32 | 68.92 |
| Max | | | 13.95 | 9.28 | 3.30 | 9.03 | 13.27 | 5.17 | 717.22 | 1.50 | 959.91 | 2.08 | 270.63 |

The evaluation results are presented in Table II. The best known values previously were calculated by four different simple heuristics [24]. We denote them as old best values.

NN→C3-Opt produced the worst results among all C3-Opt variants. Nevertheless, these results are better than the ones obtained by CIH or CIH (3-$Impr.$). The best results were obtained by Rand→C3-Opt, however, it required much more time than other approaches. The compromise variant is CIH→C3-Opt. It produces results only slightly worse than the approach with random input (0.32% versus 0.26%), but significantly outperforms it in the computational time (68.92$s$ versus 244.81$s$).

Three variants of C3-Opt (and modified CIH) improved 16 out of 18 best known values for these instances. Some improvements reached 9.68% or 13.95%, e.g., in "20_1_10" or "70_1_10". C3-Opt required more time for computation than CIH but produces better solutions. CIH→C3-Opt was able to produce solutions for instances with 30 ellipses

within $3s$, for 40 ellipses within $9s$ and for 50 ellipses within $65s$. Note, that four new best known values were obtained by involving Bisection method into the CIH. The main advantage of C3-Opt upon CIH is that it has no high "jumps" of error and evenly produces good results.

### C. Evaluation of the constricting methods

The naive constricting method could be an application of RBA algorithm after the candidate tour is constructed by either $NewTour1$ or $NewTour2$ functions, instead of the proposed constriction strategy. However, in practice, it leads fast to a local optimum. For example, for the test instances from Table I, Rand→C3-Opt obtained solutions in $38.5ms$ on the average, with an average error of 0.34% and with a maximum error of 7.29% for "$tspn2DE12\_2$". The maximum error is so high, as the method got stuck in a local optimum, caused by very "strict" constricting of a candidate tour.

## VI. CONCLUSION

Typically, an application scenario for industrial robots consists of processing work pieces with a given number of atomic tasks. Each task is typically derived from an engineering requirement. Finding optimal task sequences is critical for many important issues like processing speed or energy consumption. Today, this problem is mainly solved by programmers explicitly. Currently there exist only very few approaches, which allow automatic task sequencing. Most of them rely on simple TSP optimization. As a result, computed tours are far away from optimal solutions.

In this paper, we presented a new algorithm for very efficient computation of task sequences. The core idea is to make use of implicitly defined flexibilities and to use them for optimization. Efficiency here means both: scalable performance and near optimal results without large deviations of error. We demonstrated both effects on benchmarks taken from the optimization community. For making results comparable, we had to choose relatively simple forms (i.e., ellipses) for atomic tasks. Note, that our algorithm can be applied to other types of curves. Only the local search strategy might need to be adapted. For the same reason, we focused on optimizing Euclidean distance only. It is possible to use axis space without any method adaptation and, thus, consider PTP movements in between tasks. This would only require changes of the distance function $d$. Similarly, if a mapping from movements to energy is available, energy consumption could be minimized instead of distance.

The next step in our research will be to integrate collision-free planners in the optimization process. Although this is not a conceptual problem, it will be challenging to reach sufficient performance, as a large number of calls of a planning method will be required. Another current research direction is to allow user to enter predefined sub-sequences, which must not be changed. This extension is of great importance for many real-world applications.

## REFERENCES


[1] Z. Pan, J. Polden, N. Larkin, S. V. Duin, and J. Norrish, "Recent progress on programming methods for industrial robots," in *Proceedings for the joint conference of 41st Internationel Symposium on Robotics and 6th German Conference on Robotics*, 2010.

[2] G. Biggs and B. MacDonald, "A survey of robot programming systems," in *Proceedings of the Australasian Conference on Robotics and Automation*, 2003.

[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton University Press, 2007.

[4] S. Srivastava, S. Kumar, R. Garg, and P. Sen, "Generalized traveling salesman problem through n sets of nodes," in *CORSE Journal*, vol. 7, 1969, pp. 97–101.

[5] W. Mennell, "Heuristics for solving three routing problems: close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem," Ph.D. dissertation, University of Maryland, 2009.

[6] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell, "Touring a sequence of polygons," in *35th annual ACM symposium on Theory of Computing*. ACM Press, 2003, pp. 473–482.

[7] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics*, vol. 55, pp. 197–218, 1995.

[8] S. Arora, "Approximation schemes for NP-hard geometric optimization problems: A survey," *Mathematical Programming*, vol. 97, pp. 43–69, 2003.

[9] J. S. Mitchell, *Shortest paths and networks*. Chapman & Hall/CRC, 2004, ch. 27, pp. 607–641.

[10] D. Karapetyan and G. Gutin, "Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 208, no. 3, pp. 221–232, 2011.

[11] K. M. Elbassioni, A. V. Fishkin, and R. Sitters, "Approximation algorithms for the euclidean traveling salesman problem with discrete and continuous neighborhoods," *International Journal of Computational Geometry and Applications*, pp. 173–193, 2009.

[12] S. Alatartsev, M. Augustine, and F. Ortmeier, "Constricting Insertion Heuristic for Traveling Salesman Problem with Neighborhoods," in *23rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

[13] X. Pan, F. Li, and R. Klette, "Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons." in *Canadian Conference on Computational Geometry*, 2010, pp. 175–178.

[14] K. Baizid, R. Chellali, A. Yousnadj, A. Meddahi, and T. Bentaleb, "Genetic algorithms based method for time optimization in robotized site." in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 1359–1364.

[15] P. T. Zacharia and N. A. Aspragathos, "Optimal robot task scheduling based on genetic algorithms," *Robotics and Computer-Integrated Manufacturing*, vol. 21, pp. 67–79, 2005.

[16] I. Gentilini, F. Margot, and K. Shimada, "The travelling salesman problem with neighbourhoods: MINLP solution," *Optimization Methods and Software*, vol. 0, pp. 1–15, 2011.

[17] S. Lin, "Computer solutions of the traveling salesman problem," *The Bell System Technical Journal*, 1965.

[18] D. S. Johnson and L. A. McGeoch, *Local search in combinatorial optimization*, E. Aarts and J. K. Lenstra, Eds. John Wiley and Sons, London, 1997.

[19] F. Bock, "An algorithm for solving traveling-salesman and related network optimization problems," in *Unpublished manuscript associated with talk presented at the 14th ORSA National Meeting*, 1958.

[20] V. Mersheeva and G. Friedrich, "Routing for continuous monitoring by multiple micro UAVs in disaster scenarios," in *20th European Conference on Artificial Intelligence (ECAI)*, 2012, pp. 588–593.

[21] I. T. Hernádvölgyi, "Solving the sequential ordering problem with automatically generated lower bounds," *Operations Research Proceedings 2003*, pp. 355–362, 2003.

[22] C. Rego and F. Glover, *The Traveling Salesman Problem And Its Variations*. Kluwer Academic Publishers, 2002, ch. Local search and metaheuristics, pp. 309–368.

[23] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical recipes – the art of scientific computing*, 3rd ed. Cambridge: Cambridge University Press, 2007.

[24] S. Alatartsev, M. Augustine, and F. Ortmeier. (2012) http://euromover.cs.uni-magdeburg.de/cse/robotics/tspn/.