SDGP: A Developmental Approach for Traveling Salesman Problems

Jin Ouyang*, Thomas Weise*†, Alexandre Devert*, and Raymond Chiong‡
*Nature Inspired Computation and Applications Laboratory (NICAL),
Joint USTC-Birmingham Research Institute in Intelligent Computation and Its Applications,
School of Computer Science and Technology, University of Science and Technology of China; Hefei, Anhui, China, 230027.
†Corresponding author
§Faculty of Higher Education, Swinburne University of Technology,
50 Melba Avenue, Lilydale, Victoria 3140, Australia.

***Abstract*—This paper presents an Evolutionary Algorithm using a new ontogenic approach, called *Staged Developmental Genetic Programming* (SDGP), for solving symmetric Traveling Salesman Problems (TSPs). In SDGP, a genotype-phenotype mapping (gpm) is used to refine candidate solutions to a TSP – these candidate solutions are represented as permutations. The gpm performs several development steps, in each of which such a permutation x is incrementally modified. In each iteration within a development step, the process can choose to either apply one of seven different modifications to a specific section of x or to do nothing. The choice is made by the genotypes g, which are functions assigning a real-valued rating to the possible modifications. Smaller ratings are better and the best-rated modification is then applied, if its rating is lower than a given threshold. The genotypes are evolved using Genetic Programming in the tree-based representation well known from Symbolic Regression. Comprehensive numerical simulation experiments show that our proposed algorithm scales well with the problem size and delivers competitive results. It has an overall quadratic runtime in the number of nodes in the TSPs.**

This is a preview version of the paper [1] (see page 10 for the reference). Read the full piece at http://dx.doi.org/10.1109/CIPLS.2013.6595203.

I. Introduction

Traveling Salesman Problems (TSPs) are perhaps the most well-known logistic planning tasks. In a TSP, a vehicle has to visit a set of locations on the map and then return to its origin. The goal is to find a cyclic path of minimal costs going through all the locations [2–5]. TSPs are $\mathcal{NP}$-hard [3], meaning that it is not possible to guarantee finding the globally optimal solutions within polynomially bounded runtime. The challenge therefore, is to construct solvers that scale well, i.e., are able to quickly find good solutions even for large problem instances.

In this paper, we introduce *Staged Developmental Genetic Programming* (SDGP), an Evolutionary Algorithm (EA) applying a new ontogenic approach, for solving symmetric TSPs. In SDGP, the candidate solutions are permutations of the locations to visit. During an ontogenic genotype-phenotype mapping (gpm), such a permutation x is modified according to decisions made by a genotype g. The genotypes, which are functions evolved with Genetic Programming (GP) in the tree-representation well known from Symbolic Regression [6–8]), rate modifications that may be applied to x.

We show that SDGP produces results comparable to state-of-the-art approaches after only very few iterations. We also show that the solution quality obtained with SDGP degenerates gracefully and its runtime increases slowly with problem scale.

The remainder of this paper is organized as follows. In Section II, we briefly outline the definition of the problems (TSPs) for which SDGP is designed. After discussing related work in Section III, we introduce SDGP in Section IV and verify its properties experimentally in Section V. We conclude our paper with a discussion and plans for future work in Section VI.

II. Problem Definition

A TSP is a routing task in a street network, which can modeled as connected graph $G = (V, E)$. An instance of a symmetric TSP is defined as a tuple of

1) a set V of n vertices $v \in V$,
2) a list $E = V \times V$ of undirected edges $e = \overline{v_i\, v_j}$,
3) a function $\texttt{cost} : E \mapsto \mathbb{R}^+$, which computes the cost of traveling along an edge $e \in E$ and where $\texttt{cost}(\overline{v_i\, v_j}) = \texttt{cost}(\overline{v_j\, v_i})$, i.e., the cost is symmetric.

A candidate solution $x \in \mathbb{X}$ of a TSP can be represented as permutation of the n nodes. $x = (\texttt{A}, \texttt{B}, \texttt{C})$, for example, would mean to first visit node A, then node B, then node C, and then to return back to A, in a $n = 3$-city TSP. The objective function f, subject to minimization, is defined as follows:

$$f(x) = \texttt{cost}(\overline{x_n\, x_1}) + \sum_{i=1}^{n-1} \texttt{cost}(\overline{x_i\, x_{i+1}}) \tag{1}$$

III. Background

A. Related Work on TSPs

TSPs have been considered by researchers for more than 150 years [2]. Although they are $\mathcal{NP}$-hard, today's computing power and modern algorithms together allow us to solve instances of tens of thousands of cities [3, 9]. One of the state-of-the-art solvers is the Concorde system [9, 10]. Such algorithms involving linear programming basically can solve most instances to optimality.

However, their runtime increases steeply with the problem scale [10] and thus, they need to be massively parallelized to become feasible. Additionally, being able to solve *many* instances does not mean to be able to solve *all*. The well-known benchmark suite *TSPLIB* [11], for instance, contains the hard-to-solve instance `ts225` with only 225 cities [12].

Metaheuristic solvers are therefore being developed, on one hand, to obtain more robust optimization capabilities on one hand and, on the other hand, to provide good approximate solutions in much shorter time on the other. The aim of our work is to propose a metaheuristic approach that can quickly find such good approximations, even for large scale problems, on off-the-shelf personal computers.

To the best of our knowledge, to date there exists no other indirect representation approach to the TSP nor any application of GP[1]. Here, we briefly describe the state-of-the-art metaheuristics from the relevant literature.

Jung and Moon [13] introduced a method limited to TSPs where nodes are points on a 2D Euclidean plane. Their candidate solutions are graphical images of the tours in the plane. A curve (cut) can divide such a plane into two different areas that represent equivalence classes. A crossover operator takes the edges from parent 1 that fully fall into class 1 and those from parent 2 that full fall into class 2. This leads to a disconnected tour which is re-connected in a greedy fashion.

The $\mathcal{MAX}$-$\mathcal{MIN}$ `ACS` [14] is an example of Ant Colony Optimization (ACO) applied to TSPs. Here, simulated ants lay out pheromone on a graph depending on the length of the arcs they pass. They are more likely to follow paths with much pheromone and good heuristic values. The path of the best ant is improved by local search. This is one of the baseline algorithms for TSPs to which new methods should be compared.

The `inver-over EA` is maybe the best pure permutation-based Genetic Algorithm (GA) for TSPs. It is based on the *inver-over* search operation which is similar to one of the phenotypic update operators applied by the gpm in our approach (see Section IV-A).

Two hybrid algorithms that combine permutation-based GAs with ACO are the `hybrid-GA` [15] and the recent p-`ACGA` [16]. Whereas `hybrid-GA` utilizes a pheromone matrix to improve its crossover operator, p-`ACGA` uses ACO to mine a set of good building blocks which can be injected into a population as artificial genotypes.

Besides these notable state-of-art approaches, fundamental research regarding GAs for TSPs can be found in [17], where different search operators are assessed. This study complements our discussions in Section IV-A.

B. Related Work on Ontogenic Representations

In *indirect representations*, the search space $\mathbb{G}$ is significantly different from the solution space $\mathbb{X}$ and a genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$ translates between them. At least two classes of indirect representations may be distinguished [18, 19]: generative and ontogenic approaches. In the generative method, the gpm is a *one-shot* functional mapping from the genotypes to the phenotypes. The gpm may be an arbitrarily complex decoder, but it only uses the information given in the genotypes as input. One example for such mappings in the area of GP is Grammatical Evolution (GE) [20] with context-free grammars, where the genotypes are integer strings and the candidate solutions are sentences of a language defined by a given grammar. In GE the gpm starts with the starting symbol of that static grammar as a current variable. The first gene in the genotype identifies the first rule of the grammar fitting to the current variable to be expanded. This may result in new variables occurring, which then are expanded by rules identified by the following genes. A study on using different grammar rules to indirect encoding of Neural Network structures is given in [21, 22].

Ontogenic (or *developmental*) mappings additionally involve feedback from simulations or the process of computing the objective values when building the phenotypes in an *iterative* manner [18, 23]. Our work is a developmental, ontogenic approach that refines an existing candidate solution based on information obtained from the process of computing the objective function.

Recently, it was shown that ontogenic mappings can yield results similar to direct and generative ones but with *lesser* computational effort *despite* the more complex solution creation and evaluation process on the example of the evolution of rigid truss design optimization [18]. In a direct method, the volume of the (up to 600) beams of a truss would be optimized by a numerical optimization algorithm directly. The genotype of a generative approach could be a function that translates the coordinates of a beam to its thickness. In the ontogenic method [18], the genotypes are functions that receive as a parameter the mechanical stress on a beam and return how much the cross section of the beam should be increased. Beginning with a basic beam structure, the mechanical stress is evaluated and the function is applied to each of the beams. The updated truss is simulated again and the process is *iterated* a couple of times. The resulting structure, the phenotype, has up to 600 parameters whereas the genotypes in [18] are multilayer perceptrons representing the modification function, encoded as real vectors containing, e.g., only 12 neural weights.

IV. The SDGP Approach

The population in our new `SDGP` for symmetric TSPs consists of λ individuals p, each having a genotype $p.g$ and a phenotype $p.x$. The candidate solutions $p.x$ are node permutations discussed in Section II and obtained from the corresponding $p.g$ by applying an ontogenic gpm.

This gpm performs $\hat{\tau}$ development steps, in each of which a permutation $p.x$ is incrementally modified, as defined in Section IV-C. The initial value of $p.x$ is set to $x^\star_I$, which, in turn, is initially obtained with an initialization procedure defined in Section IV-B. In each iteration within a developmental step, `SDGP` can choose between either applying one of seven

[1]except a direct encoding scheme where the cities are terminal nodes in the program tree, which is not feasible for any non-trivial problem

different modifications (see Section IV-A) to a specific section of $p.x$ or doing nothing. The choice is made by the genotypes $p.g$, which are functions assigning a real-valued rating to each of the possible modifications. Smaller ratings are better and the best-rated modification is then applied, if its rating is lower than a given threshold. The overall SDGP procedure is defined in Section IV-D.

A. Modification Operations for $\mathbb{X}$

The search in SDGP takes place on two levels: The GP procedure searches good genotypes (mathematical functions $p.g$) which tell the gpm how to refine phenotypes $p.x$, i.e., search good permutations. For the latter case, SDGP can utilize seven different modification operations, which can be applied to candidate solutions $x = (\dots, x_i, \dots, x_j, \dots)$ for two indices $0 \leq i, j \leq n$:

- **Swap.** The swap operation $\texttt{swap}(x, i, j)$ creates a new permutation x' by swapping the nodes at index i and j, as illustrated in Figure 1.a.
- **Inversion 1.** The inversion operation $\texttt{invA}(x, i, j)$ creates a new permutation x' by inverting the sub-sequence between index i and j (inclusive) and obtains $x' = (\dots, x_{i-1}, x_j, x_{j-1}, \dots, x_{i+1}, x_i, x_{j+1}, \dots)$, as sketched in Figure 1.b. This operator is similar to the inver-over operator in [24] without adaptation.
- **Inversion 2.** Since permutations x actually represent a cyclic tour, we define $\texttt{invB}(x, i, j) \equiv \texttt{invA}(x, j, i)$, as illustrated in Figure 1.c.
- **Rotate Left 1.** The left rotation $\texttt{rolA}(x, i, j)$ creates a new permutation x' by shifting the sub-sequence between index $i + 1$ and j (inclusive) one step to the left and places x_i at index j and obtains $x' = (\dots, x_{i-1}, x_{i+1}, x_{i+2}, \dots, x_{j-1}, x_j, x_i, x_{j+1}, \dots)$, as sketched in Figure 1.d.
- **Rotate Left 2.** In Figure 1.e we sketch the second left rotation operation, defined as $\texttt{rolB}(x, i, j) \equiv \texttt{rolA}(x, j, i)$.
- **Rotate Right 1.** The right rotation $\texttt{rorA}(x, i, j)$ creates a new permutation x' by shifting the sub-sequence between index i and $j - 1$ (inclusive) one step to the right and places x_j at index i and obtains $x' = (\dots, x_{i-1}, x_{i+1}, x_{i+2}, \dots, x_{j-1}, x_j, x_i, x_{j+1}, \dots)$, as shown in Figure 1.f.
- **Rotate Right 2.** $\texttt{rorB}(x, i, j)$ is equivalent to $\texttt{rorA}(x, j, i)$ and illustrated in Figure 1.g.

We define the set Ops as $\texttt{Ops} = \{\texttt{swap}, \texttt{invA}, \texttt{invB}, \texttt{rolA}, \texttt{rolB}, \texttt{rorA}, \texttt{rorB}\}$. All operations $\texttt{op} \in \texttt{Ops}$ have the feature that $f(x')$ for their results x' when applied to candidate solution x can be computed in $\mathbf{O}(1)$ if $f(x)$ is known. This is also exemplarily illustrated in Figure 1. The application of actually carrying out the inversion or rotation operations, however, has a complexity of $\mathbf{O}(c)$ where $c = (j - i) \bmod n$.

B. Initialization

The initial best candidate solution $x_I^\star$ in SDGP is constructed in a two-step process. First, the *Double Minimum Spanning Tree* (DMST) method [25, 26] is applied. Here, a minimum spanning tree over the graph G is created by using Prim's algorithm [27] in $\mathbf{O}(n^2)$. Each node of G is visited by traveling along the edges of that tree with backtracking if a dead end is reached. All repeated visits of a node are purged. Thus, a permutation of nodes is created in time complexity $\mathbf{O}(n^2)$. If the triangle equation holds in the TSP, the total distance of this initial solution will not be worse than two times the optimal distance [28], which thus also holds for the final result of SDGP.

In the second step, exactly once for all index pairs (i, j) with $1 \leq i < j \leq n$, we iterate over the seven possible search operation applications $\texttt{op} \in \texttt{Ops}$ from the previous section. If an operation would lead to an improvement in terms of the objective function, it is immediately applied. This step also has time complexity $\mathbf{O}(n^2)$, which means in total the initialization, too, has time complexity $\mathbf{O}(n^2)$.

The initialization procedure yields a permutation $x_I^\star$ that is reasonably good in a relatively short time. It is also not too good in order to leave room for improvement and learning in the developmental gpm. $x_I^\star$ is also used as the initial guess of the best possible solution $x_B^\star$.

Algorithm 1: $(p.x, x_B^{\star\prime}) \longleftarrow \mathrm{gpm}(p.g, \hat{\tau}, x_I^\star, x_B^\star)$

```
In p.g: the genotype: a function rating search operation applications
In τ̂: the maximum number of development steps
In x_I^⋆: the initial candidate solution to be refined
InOut x_B^⋆: the overall best candidate solution ever found
Out p.x: the phenotype belonging to genotype p.g, refined from x_I^⋆
Var τ: the developmental step index
Var (i, j): the index tuple
Var op/op^⋆: the currently tested/best rated search operation
begin
    p.x ⟵ x_I^⋆
    for τ ⟵ 1 up to τ̂ do                          // perform τ̂ developmental steps
        for c ⟵ n − 2 down to 1 do                  // iterate over s index pairs
            j ⟵ τ                                   // use different pairs in each step
            for d ⟵ ⌈n/c⌉ down to 1 do               // more index pairs for small spans c
                i ⟵ 1 + [(j + 1) mod n]             // all s pairs are adjacent
                j ⟵ 1 + [(i + c) mod n]             // and have step-width c
                op^⋆ ⟵ swap                         // initialize op^⋆ (but test swap anyway)
                foreach op ∈ Ops do                 // find op^⋆ rated best by p.g among all op ∈ Ops
                    if p.g(op, p.x, i, j) < p.g(op^⋆, p.x, i, j) then   // rating?
                        op^⋆ ⟵ op                   // ...remember best rated operation
                    if f(op(p.x, i, j)) < f(x_B^⋆) then   // also: maybe new best solution?
                        x_B^⋆ ⟵ op(p.x, i, j)       // if so, remember that solution
                if p.g(op^⋆, p.x, i, j) < 0 then    // use with best-rated op iff rating < 0
                    p.x ⟵ op^⋆(p.x, i, j)           // apply best rated operation
    return (p.x, x_B^⋆)
```

C. Genotype-Phenotype Mapping

The developmental genotype-phenotype mapping gpm is the core of SDGP. It is defined in Algorithm 1 and performs $\hat{\tau}$ steps in each of which a phenotype $p.x$ may be modified by several search operation applications. $p.x$ is initialized as $x_I^\star$, the candidate solution first generated by the initialization

Fig. 1: Examples of the different operations for modifying a candidate solution x available to our system, along with efficient methods for computing the objective value $f(x')$ of the resulting new permutation x'.

We provide examples of the application of the search operations available to our system for two indices $0 \leq i < j \leq n$. The nodes are represented by literals A to I. The cost $f(x')$ of the new candidate solution x' can be computed in $\mathbf{O}(1)$ if $f(x)$ is known. Some elements of this cost update only play a role if $(j-i)n > 1$ or $(j-i) > 2$ and are marked correspondingly.

1.a: "Swap" operation: $\texttt{swap}(x, i, j)$.

$$x = (\texttt{A},\texttt{B}|\mathbf{\texttt{C}}|\texttt{D},\texttt{E},\texttt{F}|\mathbf{\texttt{G}}|\texttt{H},\texttt{I}) \Longrightarrow x' = \texttt{swap}(x, 3, 7) = (\texttt{A},\texttt{B}|\underline{\mathbf{\texttt{G}}}|\texttt{D},\texttt{E},\texttt{F}|\underline{\mathbf{\texttt{C}}}|\texttt{H},\texttt{I}) \quad (2)$$

$$f(x') = f(x) \begin{array}{l} -\texttt{cost}(\overline{\texttt{BC}}) - \texttt{cost}(\overline{\texttt{CD}}) - \texttt{cost}(\overline{\texttt{FG}}) - \texttt{cost}(\overline{\texttt{GH}}) \\ +\texttt{cost}(\overline{\texttt{BG}}) + \texttt{cost}(\overline{\texttt{GD}}) + \texttt{cost}(\overline{\texttt{FC}}) + \texttt{cost}(\overline{\texttt{CH}}) \end{array} \quad (3)$$

1.b: "Inversion" operation 1: $\texttt{invA}(x, i, j)$.

$$x = (\texttt{A},\texttt{B}|\mathbf{\texttt{C}},\texttt{D},\texttt{E},\texttt{F},\mathbf{\texttt{G}}|\texttt{H},\texttt{I}) \Longrightarrow x' = \texttt{invA}(x, 3, 7) = (\texttt{A},\texttt{B}|\underline{\mathbf{\texttt{G}}},\texttt{F},\texttt{E},\texttt{D},\underline{\mathbf{\texttt{C}}}|\texttt{H},\texttt{I}) \quad (4)$$

$$f(x') = f(x) \begin{array}{l} -\texttt{cost}(\overline{\texttt{BC}}) - \texttt{cost}(\overline{\texttt{GH}}) \\ +\texttt{cost}(\overline{\texttt{BG}}) + \texttt{cost}(\overline{\texttt{CH}}) \end{array} \quad (5)$$

1.c: "Inversion" operation 2: $\texttt{invB}(x, i, j) \equiv \texttt{invA}(x, j, i)$.

$$x = (\texttt{A},\texttt{B},\mathbf{\texttt{C}}|\texttt{D},\texttt{E},\texttt{F}|\mathbf{\texttt{G}},\texttt{H},\texttt{I}) \Longrightarrow x' = \texttt{invB}(x, 3, 7) = (\underline{\texttt{I},\texttt{H},\mathbf{\texttt{G}}}|\texttt{D},\texttt{E},\texttt{F}|\underline{\mathbf{\texttt{C}},\texttt{B},\texttt{A}}) \quad (6)$$

$$f(x') = f(x) \begin{array}{l} -\texttt{cost}(\overline{\texttt{CD}}) - \texttt{cost}(\overline{\texttt{FG}}) \\ +\texttt{cost}(\overline{\texttt{GD}}) + \texttt{cost}(\overline{\texttt{FC}}) \end{array} \quad (7)$$

1.d: "Rotate Left" operation 1: $\texttt{rolA}(x, i, j)$.

$$x = (\texttt{A},\texttt{B}|\mathbf{\texttt{C}}|\texttt{D},\texttt{E},\texttt{F},\mathbf{\texttt{G}}|\texttt{H},\texttt{I}) \Longrightarrow x' = \texttt{rolA}(x, 3, 7) = (\texttt{A},\texttt{B}|\underline{\texttt{D},\texttt{E},\texttt{F},\mathbf{\texttt{G}}|\mathbf{\texttt{C}}}|\texttt{H},\texttt{I}) \quad (8)$$

$$f(x') = f(x) \begin{array}{l} -\texttt{cost}(\overline{\texttt{BC}}) - \texttt{cost}(\overline{\texttt{CD}}) - \texttt{cost}(\overline{\texttt{GH}}) \\ +\texttt{cost}(\overline{\texttt{BD}}) + \texttt{cost}(\overline{\texttt{GC}}) + \texttt{cost}(\overline{\texttt{CH}}) \end{array} \quad (9)$$

1.e: "Rotate Left" operation 2: $\texttt{rolB}(x, i, j) \equiv \texttt{rolA}(x, j, i)$.

$$x = (\texttt{A},\texttt{B},\mathbf{\texttt{C}}|\texttt{D},\texttt{E},\texttt{F}|\mathbf{\texttt{G}}|\texttt{H},\texttt{I}) \Longrightarrow x' = \texttt{rolB}(x, 3, 7) = (\underline{\texttt{B},\mathbf{\texttt{C}}|\mathbf{\texttt{G}}}|\texttt{D},\texttt{E},\texttt{F}|\underline{\texttt{H},\texttt{I},\texttt{A}}) \quad (10)$$

$$f(x') = f(x) \begin{array}{l} -\texttt{cost}(\overline{\texttt{CD}}) - \texttt{cost}(\overline{\texttt{FG}}) - \texttt{cost}(\overline{\texttt{GH}}) \\ +\texttt{cost}(\overline{\texttt{CG}}) + \texttt{cost}(\overline{\texttt{GD}}) + \texttt{cost}(\overline{\texttt{FH}}) \end{array} \quad (11)$$

1.f: "Rotate Right" operation 1: $\texttt{rorA}(x, i, j)$.

$$x = (\texttt{A},\texttt{B}|\mathbf{\texttt{C}},\texttt{D},\texttt{E},\texttt{F}|\mathbf{\texttt{G}}|\texttt{H},\texttt{I}) \Longrightarrow x' = \texttt{rorA}(x, 3, 7) = (\texttt{A},\texttt{B}|\underline{\mathbf{\texttt{G}}|\mathbf{\texttt{C}},\texttt{D},\texttt{E},\texttt{F}}|\texttt{H},\texttt{I}) \quad (12)$$

$$f(x') = f(x) \begin{array}{l} -\texttt{cost}(\overline{\texttt{BC}}) - \texttt{cost}(\overline{\texttt{FG}}) - \texttt{cost}(\overline{\texttt{GH}}) \\ +\texttt{cost}(\overline{\texttt{BG}}) + \texttt{cost}(\overline{\texttt{GC}}) + \texttt{cost}(\overline{\texttt{FH}}) \end{array} \quad (13)$$

1.g: "Rotate Right" operation 2: $\texttt{rorB}(x, i, j) \equiv \texttt{rorA}(x, j, i)$.

$$x = (\texttt{A},\texttt{B}|\mathbf{\texttt{C}}|\texttt{D},\texttt{E},\texttt{F}|\mathbf{\texttt{G}},\texttt{H},\texttt{I}) \Longrightarrow x' = \texttt{rorB}(x, 3, 7) = (\underline{\texttt{I},\texttt{A},\texttt{B}}|\texttt{D},\texttt{E},\texttt{F}|\underline{\mathbf{\texttt{C}},\mathbf{\texttt{G}},\texttt{H}}) \quad (14)$$

$$f(x') = f(x) \begin{array}{l} -\texttt{cost}(\overline{\texttt{BC}}) - \texttt{cost}(\overline{\texttt{CD}}) - \texttt{cost}(\overline{\texttt{FG}}) \\ +\texttt{cost}(\overline{\texttt{BD}}) + \texttt{cost}(\overline{\texttt{FC}}) + \texttt{cost}(\overline{\texttt{CG}}) \end{array} \quad (15)$$

Fig. 2: Proof of quadratic complecity of gpm.

From Algorithm 1 it follows that:
$s \leq \sum_{c=1}^{n-2} \left\lceil \frac{n}{c} \right\rceil \leq \sum_{c=1}^{n-2} \left(1 + \frac{n}{c}\right) \leq n + n\sum_{c=1}^{n} \frac{1}{c}$,
i.e., $s \leq n + nH_n \approx n(1.6 + \ln n) \Rightarrow s \in \mathbf{\Theta}(n \ln n)$.

With $\text{time}(\texttt{op}) \in \mathbf{O}(c)$, the worst-case time complexity of gpm is:
$\text{time}(\text{gpm}-\text{step}) \leq \sum_{c=1}^{n-2} c\left\lceil \frac{n}{c} \right\rceil \leq \sum_{c=1}^{n-2} c\left(1 + \frac{n}{c}\right) \leq \sum_{c=1}^{n} (c+n)$,
i.e., $\text{time}(\text{gpm}-\text{step}) \leq 1.5n^2 + 0.5n \Rightarrow \text{time}(\text{gpm}-\text{step}) \in \mathbf{O}(n^2)$.

As can be seen in Figure 3.h, this is a worst-case complexity and the actual runtime seems to grow less than quadratic. The reason may be that not in all gpm-steps a modification operation is actually applied.

procedure.

The decision about which modification should be performed is made by the genotype $p.g \in \mathbb{G}$. $p.g$ therefore represents a function $p.g : \texttt{Ops} \times \mathbb{X} \times 1..n \times 1..n \mapsto \mathbb{R}$, which assigns a rating to a potential application of a search operation $\texttt{op} \in \texttt{Ops}$ to the candidate solution $p.x$ for an index pair $(i, j) : 1 \leq i, j \leq n$.

In each of the $\hat{\tau}$ steps of gpm, $s \in \mathbf{\Theta}(n \ln n)$ such index tuples are generated (see Figure 2) and all seven operations $\texttt{op} \in \texttt{Ops}$ are rated with $p.g$ for a potential application to $p.x$. Only the operation $\texttt{op}^\star$ that has received the lowest rating value (under condition $p.x$, i, and j) is considered and it is only applied if $p.g\left(\texttt{op}^\star, p.x, i, j\right)$ is less than 0 (otherwise $p.x$ is left unchanged in this iteration of the inner loop). Considering the linear complexity of the search operation application in c, gpm has a time complexity of $\mathbf{O}\left(\hat{\tau}n^2\right)$ as shown in Figure 2. This complexity is the worst-case complexity, as there may be some iterations in which no search operator is applied.

In Section IV-A we have shown that computing the new objective values $f(\texttt{op}(p.x, i, j))$ of a solution modified by operation op with parameters i and j can be done in $\mathbf{O}(1)$. As this change in objective value is one of the constants made available for $p.g$ (see Table I), it makes sense to also check whether the potential application of op would actually lead to the discovery of a new overall best solution $x_B^\star$. If so, the result of op *will* be stored in $x_B^\star$. $p.x$ is left untouched, as g may still decide to not apply op or to perform another operation. This also means that it may be sufficient if the gpm procedure touches the neighborhood of a local optimum instead of producing it directly as a result, which relaxes the search pressure to discover the exact best structure of $p.g$.

In initial, small-scale tests it turned out to be beneficial to use a non-uniform variety of different index tuples (i, j), many with i and j close to each other but also some with rather large spans $c = j - i \bmod n$. We therefore chose to create s different tuples according to a fixed scheme, where the number of index pairs with a particular span c is roughly inversely proportional c. The choice of s is mostly arbitrary, but it allows us to balance between granting more runtime to the single developmental gpms or testing more candidate solutions via parameter $\hat{\tau}$. As $\hat{\tau}$ is set independently from n, we retain quadratic complexity of the gpm.

D. Genetic Programming

In Algorithm 2, we describe the overall optimization process of SDGP. The approach starts with the initialization procedure described in Section IV-B, which yields a first guess $x_B^\star$ of the best tour. This candidate solution is also used as input $x_I^\star = x_B^\star$ for the genotype-phenotype mappings.

The population of our GP procedure holds $\lambda = 64$ individ-

Algorithm 2: The overall Genetic Programming process.

1) Obtain first candidate solution $x_B^\star = x_I^\star$ heuristically via initialization (see Section IV-B)
2) Create $\lambda = 64$ individuals p with random genotypes $p.g \in \mathbb{G}$ via Ramped-Half-and-Half and maximum tree depth 8.
3) In each of the $\hat{t} = \frac{16\,384}{\lambda * \hat{\tau}} = \frac{256}{\hat{\tau}}$ generations...
   a) For each of the $\lambda = 64$ individuals p in the population...
      i) Obtain corresponding candidate solution $p.x \in \mathbb{X}$ via genotype-phenotype mapping: $\left(p.x, x_B^\star{}'\right) = \mathrm{gpm}\left(p.g, \hat{\tau}, x_I^\star, x_B^\star\right)$.
      ii) If $f\left(x_B^\star{}'\right) < f\left(x_B^\star\right)$, set $x_B^\star = x_B^\star{}'$.
   b) Select $\mu = 16$ best individuals according to fitness v, discard the rest.
   c) Create $\lambda = 64$ new individuals (maximum tree depth 8) by using
      i) sub-tree exchange crossover (with probability 0.25).
      ii) sub-tree replacement mutation (with probability 0.75).
   d) Set $x_I^\star = x_B^\star$.
4) Return best solution $x_B^\star$ ever encountered.

uals p, which evolve according to a (μ, λ) scheme (μ being 16). Each individual holds a genotype $p.g \in \mathbb{G}$, which is a mathematical function in the well-known tree representation from symbolic regression [6–8]. These trees have a maximum depth of 8 and are composed of the functions and terminals given in Table I. As discussed in Section IV-C, these genotypes drive a gpm that iteratively refines $x_I^\star$ and yields the phenotype $p.x$ corresponding to each function $p.g$. The gpm, in a loop, presents several index pairs (i, j) to the genotype for each search operation $\mathtt{op} \in \mathtt{Ops}$ (see Section IV-A), along with the current $p.x$. By using the six approach-specific terminal symbols given in Table I, the genotypes rate these potential applications and the best-rated one $\mathtt{op}^\star$ is applied if its rating is below 0. At each step during each gpm, it may be possible to also discover a new best tour $x_B^\star$.

After all phenotypes have been built, the $\mu = 16$ best individuals are selected. Several small scale experiments have revealed that using the fitness function v given in Equation 22, which incorporates both the (normalized) objective value $f(p.x)$ of the phenotype $p.x$ belonging to a genotype $p.g$, and the total number of search operations applied, leads to the best results.

$$v(p.x) = \frac{f(p.x)}{f(x_I^\star)} + \frac{1}{\text{total number of search operator applications}} \quad (22)$$

At the end of each generation, we also set $x_I^\star = x_B^\star$. In other words, the input candidate solution to the genotype-phenotype mapping is always the best solution known *before* the current generation. This is somewhat similar to the staged development approach in our previous work [18]. It leads to a salient improvement compared to always using the same solution (produced by the initialization procedure).

If a new best solution $x_B^\star$ is discovered, this is achieved by a genotype modifying the current initial solution $x_I^\star$ in a way so that at least one candidate solution neighboring $x_B^\star$ is generated during the gpm.[2]. As the gpm receives the same inputs $x_I^\star$ for each genotype $p.g$, finding the same new (local) optimum would mean they make the same modifications to $x_I^\star$. By setting $x_I^\star = x_B^\star$ at the end of each generation, however, this is not necessary. Instead, $x_B^\star$ becomes accessible to all individuals in the next generation. Additionally, the modifications leading to $x_B^\star$ are no longer required in any genotype and can be pruned by the search, literally making room for new information, more decisions, and adaptions to the new situation.

The overall time complexity of our method evaluates to $\mathbf{O}\left(\lambda \hat{t} \hat{\tau} n^2\right)$ and is the sum of the complexity $\mathbf{O}\left(n^2\right)$ of the initialization procedure (see Section IV-B) and $\lambda \hat{t}$ times the complexity $\mathbf{O}\left(\hat{\tau} n^2\right)$ of the gpm, where $\hat{t}$ is the maximum number of generations to perform.[3]. As $\hat{\tau}$, $\hat{t}$, and λ can be set as constants independently of the number n of nodes of the TSP, this results in an overall complexity of $\mathbf{O}\left(n^2\right)$.

TABLE I: The function and terminal set of the Genetic Programming.

Approach-Specific Terminals: same value for each $\mathtt{op} \in \mathtt{Ops}$

$\mathtt{T_{DS}}$. A terminal $\mathtt{T_{DS}}$ giving normalized information about index τ of the current *development step*.

$$\mathtt{T_{DS}} = \frac{\tau + 0.5}{\hat{\tau}} \quad (16)$$

$\mathtt{T_{BI}}$. The scaled *best improvement* that could potentially be achieved with *any* search operation for the given candidate solution $p.x$ and index tuple (i, j). Negative values are good.

$$\mathtt{T_{BI}} = \frac{1}{f(x_I^\star)} \arg\min_{\mathtt{op}' \in \mathtt{Ops}} \left\{ f\left(\mathtt{op}'(p.x, i, j)\right) - f(p.x) \right\} \quad (17)$$

$\mathtt{T_{CI}}$. The scaled cost difference of the current candidate solution $p.x$ to the initial candidate solution $x_I^\star$, i.e., the *current improvement*. Negative values are good.

$$\mathtt{T_{CI}} = \frac{1}{f(x_I^\star)} \left(f(p.x) - f(x_I^\star) \right) \quad (18)$$

Approach-Specific Terminals: different value for each $\mathtt{op} \in \mathtt{Ops}$

$\mathtt{T_{OI}}$. The scaled cost improvement that the currently rated operator $\mathtt{op}$ can provide for candidate solution $p.x$ and index tuple (i, j). Negative values are good.

$$\mathtt{T_{OI}} = \frac{1}{f(x_I^\star)} \left(f(\mathtt{op}(p.x, i, j)) - f(p.x) \right) \quad (19)$$

$\mathtt{T_{RF}}$. The relative frequency $\mathtt{T_{RF}}$ with which operator $\mathtt{op}$ was applied (selected by $p.g$) during the current call to gpm.

$$\mathtt{T_{RF}} = \frac{\text{number of applications of } \mathtt{op}}{\max\{1, \text{total number of all search operator applications}\}} \quad (20)$$

$\mathtt{T_{MD}}$. The mean difference $\mathtt{T_{MD}}$ in terms of f that the past applications of this operator made during the current call to gpm. Zero if operator was not applied yet. Negative values are good.

$$\mathtt{T_{MD}} = \frac{\sum_{\forall \text{applications of } \mathtt{op}} (f(p.x \text{ after application}) - f(p.x \text{ before application}))}{\max\{1, \text{number of applications of } \mathtt{op}\}} \quad (21)$$

Standard Functions and Terminals

$+$, $-$, $*$, $\mathbf{e^a}$, $|\cdot|$, $\sin$. Unprotected arithmetic operators, all results and parameters are possible.

$/$, $\sqrt{\cdot}$. Protected operators arithmetic: NaN maps to 1, $\sqrt{a}$ maps to $-\sqrt{-a}$ for $a < 0$.

ERC. Ephemeral random constant [6]: initialized with random number uniformly distributed in $[0, 1)$.

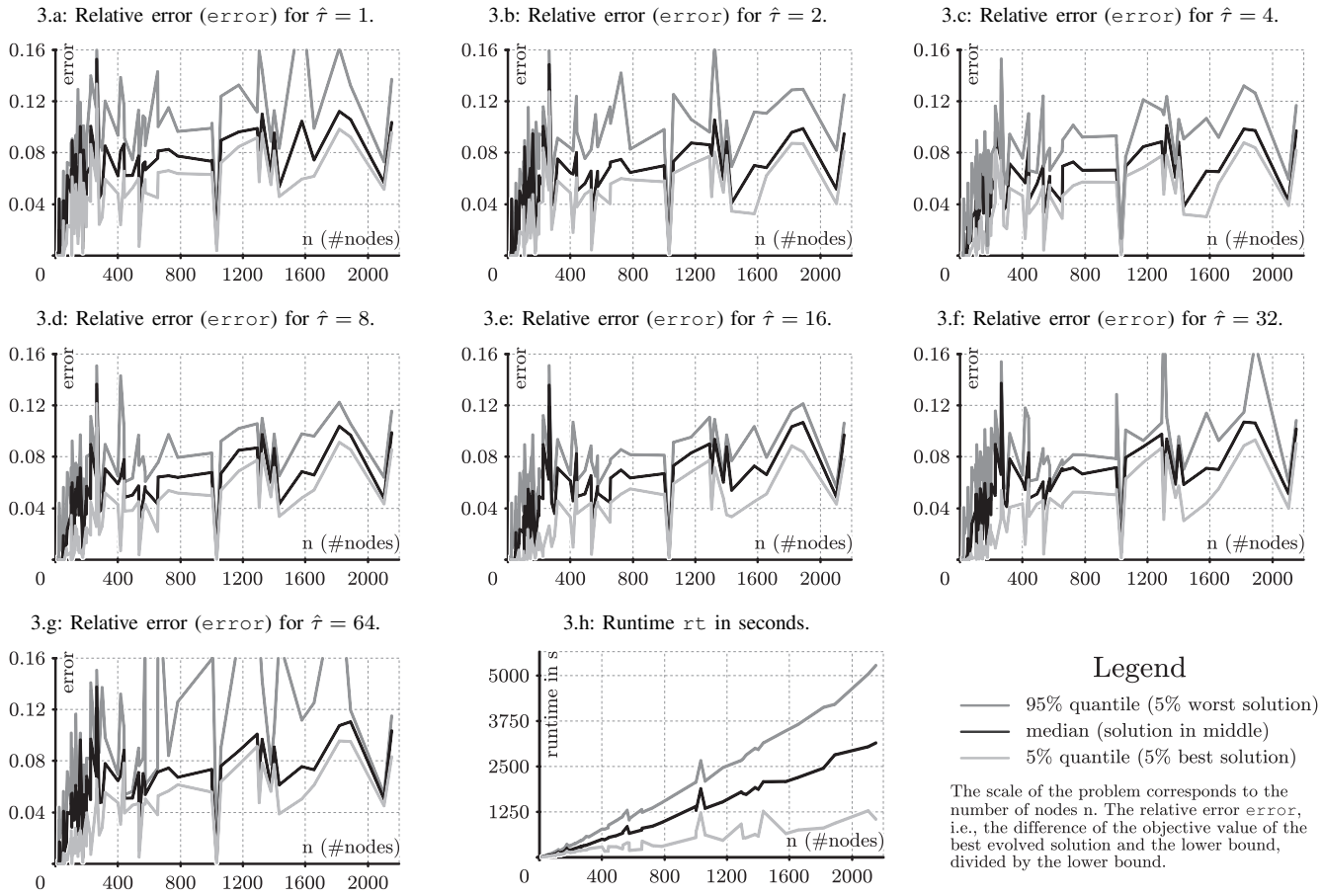
V. Experiments

With our experiments, we aim to answer the following three research questions:

[2] $x_B^\star$ does not necessarily need to be directly traversed during the gpm, as outlined in Algorithm 1.

[3] The operations modifying the *genotypes* (trees) have time complexity $\mathbf{O}(1)$, as the tree size is limited by a constant (maximum depth 8).

Fig. 3: Relative error error and runtime of SDGP in relation to the problem scale (number of nodes n).



1) How does SDGP scale with the number n of nodes in TSPs?
2) How does its performance compare with other state-of-the-art metaheuristics?
3) Which setting of $\hat{\tau}$ is good if the same total number of development steps for the whole optimization process is granted? Smaller values of $\hat{\tau}$ give less time to the developmental gpm but allow for more generations of GP, whereas higher values decrease the total number of candidate solutions produced but allow for longer refinement cycles.

In our experiments, we performed 30 runs on 85 symmetric TSP instances from the well-known *TSPLIB* [11] benchmark suite. SDGP was implemented in Java 1.7 and run on an Intel Core i3-2120 CPU with 3.3GHz and 4 GB RAM with Windows 7 and Java 1.7.0_03.

We applied GP in a (μ, λ)-fashion with $\lambda = 64$ and $\mu = 16$ and set the number of generations to $\hat{t} = \frac{256}{\hat{\tau}}$. We tested different numbers of development steps per gpm: $\hat{\tau} = 2^i$ for all $i \in 0..6$. The total number of development steps is thus fixed at $256 * 64 = 16\,384$.

In terms of the solution quality, we have used the difference between the objective value of the best candidate solution $x^\star_B$ found in a run and the objective value $\breve{f}$ of the known global optimum, divided by $\breve{f}$, that is the relative error $\texttt{error}(x^\star_B) = \frac{f(x^\star_B) - \breve{f}}{\breve{f}}$. In Figure 3, we plot statistics on error for different values of $\hat{\tau}$ over the problem scale n. Figures 3.a to 3.g all show similar characteristics, meaning that SDGP is robust in terms of different $\hat{\tau}$ settings. error increases slowly, usually staying below 8% in median with the top-5% solutions often coming close to 0. The solution quality provided by SDGP for a fixed budget of development steps degenerates gracefully with the problem scale n. The time per run rt in seconds over all settings of $\hat{\tau}$ is plotted in Figure 3.h. From there and Table II can be seen that it grows slowly with n. In a nutshell, we find that SDGP scales well.

The best setting for $\hat{\tau}$ seems to be 16 (see Figure 3.e). Here, the 95% and 5% quantile of error are closest together, i.e., the optimization process is robust. If we apply a two-tailed Mann-Whitney U test with significance level 0.02 to compare the seven $\hat{\tau}$ settings over all 85 benchmark instances, we find that $\hat{\tau} = 16$ outperforms 201 comparisons, loses 12, and does not perform significantly different in 297. The second best setting is $\hat{\tau} = 8$, which can win 187 comparisons and loses 11.

In Table II, we list the objective values, mean, and median

TABLE II: Selected results of the $\hat{\tau} = 16$ experiments in comparison with other methods. The first three columns list the features of the benchmark instances. Next we provide the objective values of the best, mean, and median solutions resulting from 30 runs of SDGP and give the median runtime rt measured in seconds over these runs (different from Figure 3.h which is over all $\hat{\tau}$ settings). The last six comments list results taken from the literature cited in the columen headers.

TSP Instance Features			Statistics on SDGP				p-ACGA [16]		EA [24]	[15]		[14]
instance	n	$\check{f}$	best	mean	median	med. rt	best	mean	mean	best	mean	mean
eil51	51	426	**426**	434.3	435.0	36.7	427	430.3	**426.0**	428	428.5	
berlin52	52	7542	**7542**	7668.4	7542.0	36.9	**7542**	**7615.4**				
st70	70	675	680	683.4	683.0	56.0			**675.0**			
eil76	76	538	**538**	555.2	557.5	54.9	**538**	548.4	**538.0**			
kroa100	100	21 282	**21 282**	22 223.6	22 319.5	85.0			**21 282.0**	21 285	21 285.0	21 427.0
krob100	100	22 141	**22 141**	22 623.1	22 527.5	79.2	22 179	**22 510.1**				
kroc100	100	20 749	**20 749**	21 460.5	21 652.0	84.3	**20 749**	21 064.6	**20 749.0**			
krod100	100	21 294	**21 294**	21 851.7	21 803.5	82.5	21 330	21 779.1	**21 294.0**			
kroe100	100	22 068	**22 068**	22 689.4	22 608.5	82.6	22 121	**22 374.6**				
rd100	100	7910	**7910**	8223.1	8269.0	71.6	**7910**	**8044.3**				
eil101	101	629	632	651.7	647.5	89.8	631	641.5	**629.2**			
lin105	105	14 379	**14 379**	14 480.0	14 401.0	82.5	**14 379**	14 574.5	**14 379.0**			
bier127	127	118 282	120 853	122 128.4	122 375.0	118.6	**118 695**	**120 377.6**				
ch130	130	6110	**6137**	6392.3	6445.0	121.9	**6137**	**6277.9**				
ch150	150	6528	6584	6791.1	6826.0	133.2	**6549**	**6646.6**				
kroa150	150	26 524	26 792	**27 200.3**	27 231.0	133.3	**26 714**	27 302.9				
krob150	150	26 130	**26 187**	27 484.4	27 517.0	135.0	26 310	**26 760.7**				
d198	198	15 780	15 891	16 281.8	16 321.5	161.2				**15 797**	**15 839.5**	15 856.0
kroa200	200	29 368	29 868	30 721.7	30 624.5	177.1	**29 471**	**30 118.8**				
krob200	200	29 437	30 189	31 076.1	31 371.5	185.6	**29 743**	**30 366.0**				
pr299	299	48 191	49 059	**50 017.2**	49 923.5	300.4	**48 995**	50 019.1				
lin318	318	42 029	43 349	44 647.6	44 483.5	325.4	42 820	43 550.1		**42 334**	42 605.3	**42 426.0**
pcb442	442	50 778	52 530	53 239.8	53 275.0	525.6	52 263	53 719.8	**51 097.5**			51 794.0
att532	532	27 686	28 765	29 524.4	29 495.5	640.2						**28 233.0**
rat575	575	6773	**7062**	**7118.5**	7122.5	632.5	7081	7152.0				
rat783	783	8806	9286	9369.5	9366.5	886.2	9235	9374.2				**9142.0**
pr1002	1002	259 045	**272 172**	**275 672.2**	275 804.0	1149.0	274 828	277 906.7				
pcb1173	1173	56 892	**60 496**	61 610.8	61 620.0	1413.5	60 910	**61 184.0**				
rl1323	1323	270 199	**289 545**	**294 883.3**	295 508.5	1634.7	294 547	296 999.4				
fl1400	1400	20 127	**20 761**	21 811.1	21 893.0	1716.1	21 037	**21 117.8**				
d1655	1655	62 128	**65 138**	66 270.9	66 243.5	1961.4	65 484	**66 078.4**				
u1817	1817	57 201	62 138	63 132.2	63 122.5	2305.3						
rl1889	1889	316 536	342 947	349 416.3	350 347.0	2381.0						
d2103	2103	79 952	83 363	83 906.7	83 851.0	2919.0						
u2152	2152	64 253	68 786	70 449.2	70 492.5	2913.6						

result quality obtained with setting $\hat{\tau} = 16$ for selected benchmark instances over 30 runs. We also provide the median of the runtime rt obtained with this configuration (as opposed to Figure 3.h, where the rt statistics are given over all $\hat{\tau}$ settings). We find that SDGP is not fast for small problem instances. However, rt increases very slowly with the problem scale, only slightly super-linear (showing again that the quadratic time complexity is a worst-case issue). Here rt often tends to be close to the number n of nodes, just in seconds, for small problems and close to $1.5n$ for larger ones.

Comparing the results of SDGP with the presented settings to results of other methods published in the literature is not easy. The reason is that these usually are obtained with different termination criteria (and hence significantly more iterations), with algorithms that have a time complexity in at least $\mathbf{O}(n^3)$, for different numbers of runs, and often are incomplete.

We still list the results of the approaches listed as related work in Section III-A for the sake of completeness. SDGP is comparable to p-ACGA [16], in terms of its performance. However, the GA part of p-ACGA runs for $50n$ generations with population size of 100, which avails to, e.g., $50 * 51 * 100 = 255\,000$ evaluated candidate solutions for the small-scale problem *eil51*, as opposed to the 16 384 development steps in total used by SDGP for each problem. Still, SDGP is generally on par with this method, sometimes providing better best solutions, sometimes worse.

The inver-over EA can be said to outperform both methods. However, the results reported in [24] stem from experiments where this EA was run until stagnation. For example, the number of applied search operations for the small-scale problem *eil51* were reported as 147 972. Our goal was to find good solutions fast, within time complexity of $\mathbf{O}(n^2)$.

The hybrid-GA [15] was run for 1000 generations with n individuals, i.e., also for the *eil51* problem the results are based on at least 51 000 evaluated candidate solutions. Only results for four instances are reported, in two of which SDGP finds better best solutions. The results of $\mathcal{MAX}$-$\mathcal{MIN}$ ACS [14] were obtained by performing $100n$ steps with complexity

$\mathbf{O}(n^2)$, i.e., in $\mathbf{O}(n^3)$. Hence this system exhibits better mean performance than SDGP.

The results provided by SDGP in our experiments are not better than those of the related state-of-the-art methods. However, our results were obtained with much fewer iterations, so they are actually very encouraging. From 54 statistical values which can be compared with p−ACGA (which uses significantly more iterations), it is better in 17 and not different in 6, for example.

VI. Conclusions

In this paper we introduced SDGP, an indirect encoding method for solving TSPs. As optimization algorithm, we have applied GP where the genotypes are mathematical functions in the common tree encoding. The phenotypes are permutations representing the node sequences, i.e., the usual candidate solutions for TSPs. The first such candidate solution is created by an initialization procedure. In a genotype-phenotype mapping, the genotypes select potential modification operations to be applied to the best phenotype known before the current generation in order to refine it and to discover better results.

We have shown that SDGP has an overall quadratic time complexity. Through extensive simulation experiments, we found that the solution quality provided by it decreases slowly with the problem scale n and its runtime increases slowly as well.

It is not easy to compare our approach to results reported in the literature, as these have been obtained with largely different setups, in particular usually by granting the respective algorithms significantly more iterations. As part of our future work, it is therefore necessary to conduct experiments with different algorithms in order to create a more sound basis for algorithm comparisons.

Acknowledgements. This work has been supported by the National Natural Science Foundation of China (NSFC) Grant Number 61150110488, the China Postdoctoral Science Foundation Special Financial Grant number 201104329, and the Chinese Academy of Sciences (CAS) Fellowship for Young International Scientists 2011Y1GB01.

References

[1] J. Ouyang, T. Weise, A. Devert, and R. Chiong, "SDGP: A Developmental Approach for Traveling Salesman Problems," in *Proceedings of the 2013 IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS'13)*. Singapore: Grand Copthorne Waterfront Hotel: Los Alamitos, CA, USA: IEEE Computer Society Press, April 15–19, 2013, pp. 78–85.

[2] B. F. Voigt, *Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur*. Ilmenau, Germany: Voigt, 1832, excerpt: "… Durch geeignete Auswahl und Planung der Tour kann man oft so viel Zeit sparen, daß wir einige Vorschläge zu machen haben. … Der wichtigste Aspekt ist, so viele Orte wie möglich zu erreichen, ohne einen Ort zweimal zu besuchen. …".

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, ser. Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, February 2007. [Online]. Available: http://books.google.de/books?id=nmF4rVNJMVsC

[4] F. Greco, Ed., *Traveling Salesman Problem*. Vienna, Austria: IN-TECH Education and Publishing, September 2008. [Online]. Available: http://intechweb.org/downloadfinal.php?is=978-953-7619-10-7&type=B

[5] E. L. G. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, ser. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, UK: Wiley Interscience, September 1985. [Online]. Available: http://books.google.de/books?id=BXBGAAAAYAAJ

[6] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, ser. Bradford Books. Cambridge, MA, USA: MIT Press, December 1992, 1992 first edition, 1993 second edition. [Online]. Available: http://books.google.de/books?id=Bhtxo60BV0EC

[7] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. London, UK: Lulu Enterprises UK Ltd, March 2008, with contributions by John R. Koza. [Online]. Available: http://www.lulu.com/items/volume_63/2167000/2167025/2/print/book.pdf

[8] T. Weise, *Global Optimization Algorithms – Theory and Application*. Germany: it-weise.de (self-published), 2009. [Online]. Available: http://www.it-weise.de/projects/book.pdf

[9] W. J. Cook, D. G. Espinoza, and M. Goycoolea, "Computing with Domino-Parity Inequalities for the TSP," Atlanta, GA, USA: Georgia Institute of Technology, Industrial and Systems Engineering, Tech. Rep., 2005. [Online]. Available: http://www2.isye.gatech.edu/~wcook/papers/DP_paper.pdf

[10] W. J. Cook, "Results of Concorde for TSPLib Benchmark," December 2003. [Online]. Available: http://www.tsp.gatech.edu/concorde/benchmarks/bench99.html

[11] G. Reinelt, "TSPLIB," 1995. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[12] V. Chvátal, "The Traveling Salesman Problem," October 2008. [Online]. Available: http://users.encs.concordia.ca/~chvatal/tsp/tsp.html

[13] K. Jung and B. Moon, "Toward Minimal Restriction of Genetic Encoding and Crossovers for the Two-Dimensional Euclidean TSP," vol. 6, no. 6, pp. 557–565, December 2002. [Online]. Available: 10.1109/TEVC.2002.804321

[14] T. Stützle and H. H. Hoos, "MAX-MIN Ant System and Local Search for the Traveling Salesman Problem," in *Proceedings of the IEEE International Conference on Evolutionary Computation (CEC'97)*, T. Bäck, Z. Michalewicz, and X. Yao, Eds. Indianapolis, IN, USA: Piscataway, NJ, USA: IEEE Computer Society, April 13–16, 1997, pp. 309–314. [Online]. Available: http://www.gta.ufrj.br/ensino/cpe717-2011/stutzle97-icec.pdf

[15] C. M. White and G. G. Yen, "A Hybrid Evolutionary Algorithm for Traveling Salesman Problem," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'04)*, vol. 2. Portland, OR, USA: Los Alamitos, CA, USA: IEEE Computer Society Press, June 20–23, 2004, pp. 1473–1478.

[16] P. Chang, W. Huang, and Z. Zhang, "A Puzzle-Based Genetic Algorithm with Block Mining and Recombination Heuristic for the Traveling Salesman Problem," *Journal of Computer Science and Technology (JCST)*, vol. 27, no. 5, pp. 937–949, September 5, 2012. [Online]. Available: http://jcst.ict.ac.cn:8080/jcst/EN/10.1007/s11390-012-1275-3

[17] X. Yao, "An Empirical Study of Genetic Operators in Genetic Algorithms," *Microprocessing and Microprogramming*, vol. 38, no. 1-5, pp. 707–714, September 1993. [Online]. Available: http://www.cs.bham.ac.uk/~xin/papers/euro93_final.pdf

[18] A. Devert, T. Weise, and K. Tang, "A Study on Scalable Representations for Evolutionary Optimization of Ground Structures," *Evolutionary Computation*, vol. 20, no. 3, pp. 453–472, Fall 2012. [Online]. Available: http://www.marmakoide.org/download/publications/devweita-ecj-preprint.pdf

[19] P. J. Bentley and S. P. Kumar, "The Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, W. Banzhaf, J. M. Daida, Á. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and R. E. Smith, Eds. Orlando, FL, USA: San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., July 13–17, 1999, pp. 35–43. [Online]. Available: http://www.cs.ucl.ac.uk/staff/ucacpjb/BEKUC1.pdf

[20] C. Ryan, J. J. Collins, and M. O'Neill, "Grammatical Evolution: Evolving Programs for an Arbitrary Language," in *Proceedings of the First European Workshop on Genetic Programming (EuroGP'98)*, ser. Lecture Notes in Computer Science (LNCS), W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds., vol. 1391/1998. Paris, France: Berlin, Germany: Springer-Verlag GmbH, April 14–15, 1998, pp. 83–95. [Online]. Available: http://www.grammatical-evolution.org/papers/eurogp98.ps

[21] X. Yao and Y. Shi, "A Preliminary Study on Designing Artificial Neural Networks Using Co-Evolution," in *Proceedings of 1st IEEE Singapore International Conference on Intelligent Control and Instrumentation (SICICI'95)*. Singapore: IEEE (Institute of Electrical and Electronics Engineers), IEEE Singapore Section, July 2–8, 1995, pp. 149–154, invited Paper. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.6071

[22] X. Yao, "Evolving Artificial Neural Networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, September 1999, invited Paper. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.793

[23] A. Devert, "Building Processes Optimization: Toward an Artificial Ontogeny based Approach," Ph.D. dissertation, Paris, France: Université Paris-Sud, Ecole Doctorale d'Informatique and Orsay, France: Institut National de Recherche en Informatique et en Automatique (INRIA), Centre de Recherche Saclay – Île-de-France, May 2009.

[24] G. Tao and Z. Michalewicz, "Inver-over Operator for the TSP," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, ser. Lecture Notes in Computer Science (LNCS), Á. E. Eiben, T. Bäck, M. Schoenauer, and H. Schwefel, Eds., vol. 1498/1998. Amsterdam, The Netherlands: Berlin, Germany: Springer-Verlag GmbH, September 27–30, 1998, pp. 803–812. [Online]. Available: http://cs.adelaide.edu.au/~zbyszek/Papers/p44.pdf

[25] D. S. Johnson and L. A. McGeoch, "Experimental Analysis of Heuristics for the STSP," in *The Traveling Salesman Problem and its Variations*, ser. Combinatorial Optimization, G. Z. Gutin and A. P. Punnen, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002, vol. 12, ch. 9, pp. 369–443. [Online]. Available: http://www2.research.att.com/~dsj/papers/stspchap.pdf

[26] N. Christofides, "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem," Pittsburgh, PA, USA: Carnegy Mellon University (CMU), Graduate School of Industrial Administration, Management Sciences Research Group, Management Sciences Research Report 388 / CS-93-13, February 1976.

[27] R. C. Prim, "Shortest Connection Networks and Some Generalizations," *Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, November 1957. [Online]. Available: www.alcatel-lucent.com/bstj/vol36-1957/articles/bstj36-6-1389.pdf

[28] V. Jarník, "O Jistém Problému Minimálním: (Z Dopisu Panu O. Borůskovi)," *Práce Moravské Přírodovědecké Společnosti: Acta Societatis Scientiarum Naturalium Moravia*, vol. 6, pp. 57–63, 1930. [Online]. Available: http://books.google.de/books?id=GOc3HAAACAAJ

[29] T. Weise, R. Chiong, K. Tang, J. Lässig, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao, "Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem," *IEEE Computational Intelligence Magazine (CIM)*, vol. 9, no. 3, pp. 40–52, August 2014.

```
@inproceedings{OWDC2013SADAFTSP,
  author    = {Jin Ouyang and Thomas Weise and Alexandre Devert
               and Raymond Chiong},
  title     = {{SDGP: A Developmental Approach for Traveling
               Salesman Problems}},
  booktitle = {Proceedings of the 2013 IEEE Symposium on
               Computational Intelligence in Production and
               Logistics Systems (CIPLS'13)},
  publisher = {Los Alamitos, CA, USA: IEEE Computer Society
               Press},
  address   = {Singapore: Grand Copthorne Waterfront Hotel},
  pages     = {78--85},
  year      = {2013},
  month     = apr # {~15--19, },
  doi       = {10.1109/CIPLS.2013.6595203},
  eiid      = {20134116837899},
  inspec    = {13752116},
},
```


After this paper, we began to more seriously focus on benchmarking TSP algorithms and presented *TSP Suite*, a holistic framework for TSP solver development, testing, benchmarking, and comparison in [29]. Our results there showed that the algorithm presented here (in this current paper) is actually not good at all.