Evolutionary Algorithms and Dynamic Programming

Benjamin Doerr[a], Anton Eremeev[b], Frank Neumann[c], Madeleine Theile[d], Christian Thyssen[e,1]

[a]*Algorithms and Complexity, Max-Planck-Institut für Informatik, Saarbrücken, Germany*
[b]*Omsk Branch of Sobolev Institute of Mathematics SB RAS, Omsk, Russia*
[c]*School of Computer Science, University of Adelaide, Adelaide, Australia*
[d]*Institut für Mathematik, TU Berlin, Berlin, Germany*
[e]*Fakultät für Informatik, LS 2, TU Dortmund, Dortmund, Germany*

Abstract

Recently, it has been proven that evolutionary algorithms produce good results for a wide range of combinatorial optimization problems. Some of the considered problems are tackled by evolutionary algorithms that use a representation which enables them to construct solutions in a dynamic programming fashion. We take a general approach and relate the construction of such algorithms to the development of algorithms using dynamic programming techniques. Thereby, we give general guidelines on how to develop evolutionary algorithms that have the additional ability of carrying out dynamic programming steps. Finally, we show that for a wide class of the so-called DP-benevolent problems (which are known to admit FPTAS) there exists a fully polynomial-time randomized approximation scheme based on an evolutionary algorithm.

Keywords: Combinatorial optimization, dynamic programming, evolutionary algorithms, approximation algorithms

1. Introduction

Evolutionary algorithms (EAs) [14] have been shown to be successful for a wide range of optimization problems. While these algorithms work well for many optimization problems in practice, a satisfying and rigorous

---

[1]né Horoba

mathematical understanding of their performance is an important challenge in the area of evolutionary computing [1].

Interesting results on the runtime behaviour of evolutionary algorithms have been obtained for a wide range of combinatorial optimization problems (see [34] for a comprehensive presentation). This includes well-known problems such as sorting and shortest paths [39], spanning trees [33], maximum matchings [19], and minimum cuts [30, 31]. There are also some results on evolutionary algorithms acting as approximation algorithms for NP-hard problems like partition [42], covering [17], and multi-objective shortest path [22, 32] problems. But a general theoretical explanation of the behavior of evolutionary algorithms is still missing. The first step in this direction is taken in [37], where the authors show for an important subclass of optimization problems that evolutionary algorithms permit optimal solutions in polynomial time.

1.1. Main Contributions

The aim of this paper is to make another contribution to the theoretical understanding of evolutionary algorithms for combinatorial optimization problems. We focus on the question how to represent possible solutions such that the search process becomes provably efficient. When designing an evolutionary algorithm for a given problem, a key question is how to choose a good representation of possible solutions. This problem has been extensively studied in the literature on evolutionary algorithms [38]; for example there are different representations for the well-known traveling salesman problem (see e. g. Michalewicz [27]) or NP-hard spanning tree problems (see e. g. Raidl and Julstrom [36]).

Each of these representations induces a different neighborhood of a particular solution, and variation operators such as mutation and crossover have to be adjusted to the considered representation. Usually, such representations either lead directly to feasible solutions for the problem to be optimized or the search process is guided towards valid solutions by using some penalty functions. Here, the representation of possible solutions in combination with some suitable variation operators may be crucial for the success of the algorithm.

Recently, it has been proven for various combinatorial optimization problems that they can be solved by evolutionary algorithms in reasonable time using a suitable representation together with mutation operators adjusted to the given problem. Examples for this approach are the single source shortest

path problem [39], all-pairs shortest path problem [9], multi-objective shortest path problem [22], the travelling salesman problem [41] and the knapsack problem [15]. The representations used in these papers are different from the general encodings working with binary strings as considered earlier in theoretical works on the runtime behavior of evolutionary algorithms. Instead, the chosen representations reflect some properties of partial solutions of the problem at hand that allow to obtain solutions that can be extended to optimal ones for the considered problem. To obtain such partial solutions the algorithms make use of certain diversity mechanisms allowing the algorithms to proceed in a dynamic programming way.

Note that the problem-solving capability of classical genetic algorithms is sometimes explained using the building block hypothesis [20], which also involves extension of partial solutions to the optimal ones. A relation of the mentioned above EAs to dynamic programming, however, allows to obtain more specific results in terms of average optimization time.

Dynamic programming (DP) [3] is a well-known algorithmic technique that helps to tackle a wide range of problems. A general framework for dynamic programming has been considered by e.g. Woeginger [43] and Klötzler [25]. The technique allows to compute an optimal solution for the problem by extending partial solutions to an optimal one.

An important common feature of the evolutionary algorithms [9, 13, 15, 22, 39, 41] is that each of them is based on a suitable multi-objective formulation of the given problem. The schemes of these EAs and solution representations are different, however.

The algorithms proposed in [39] and [15] are generalizations of the well-known (1+1)-EA (see e.g. [4]) to the multi-objective case and they are based on a different representation of solutions than the one used in our paper.

The $(\mu + 1)$-EA in [41] employs a large population of individuals, where each individual encodes just one partial solution. In [9, 13] it was shown that, for the all-pairs shortest path problem on an n-vertex graph, application of a suitable crossover operator can provably reduce the optimization time of the EA by a factor of almost $n^{3/4}$.

A special case of the DP-based evolutionary algorithm proposed in the present paper can be found, e.g. in [22]. Both algorithms employ large populations of individuals where an individual encodes a partial solution. The outline of these algorithms is similar to that of the SEMO algorithm [26].

Each gene in our problem representation defines one of the DP transition mappings, and a composition of these mappings yields the DP state rep-

3

resented by the individual. The proposed EA utilizes a mutation operator which is a special case of point mutation, where the gene subject to change is not chosen randomly as usual, but selected as the first gene which has never been mutated so far (see Section 3.2 for details and links to the biological systems).

The goal of the paper is to relate the above mentioned multi-objective evolutionary approaches to dynamic programming and give a general setup for evolutionary algorithms that are provably able to solve problems having a dynamic programming formulation. In particular, we show that in many cases a problem that can be solved by dynamic programming in time T has an evolutionary algorithm which solves it in expected time $O(T \cdot n \cdot \log(|DP|))$ with n being the number of phases and $|DP|$ being the number of states produced at the completion of dynamic programming.

The obtained results are not aimed at the development of faster solution methods for the combinatorial optimization problems (to construct an EA in our framework, one has to know enough about the problem so that the traditional DP algorithm could be applied and this algorithm would be more efficient). Instead, we aim at characterizing the area where evolutionary algorithms can work efficiently and study the conditions that ensure this. To put it informally, our results imply that a class of problems that is easy for the DP algorithm is also easy for a suitable EA for most of the reasonable meanings of the term "easy" (solvable in polynomial or pseudo-polynomial running time or admitting FPTAS).

1.2. Organization

The rest of the paper is organized as follows. In Section 2, we introduce a general dynamic programming formulation and the kind of problems that we want to tackle. This dynamic programming approach is transferred into an evolutionary algorithm framework in Section 3. Here we also show how to obtain evolutionary algorithms carrying out dynamic programming for some well-known combinatorial optimization problems. In Section 4, we consider a wide class of the DP-benevolent problems which are known to have fully polynomial-time approximation schemes based on dynamic programming [43]. We show that for the problems of this class there exists a fully polynomial-time randomized approximation scheme based on an evolutionary algorithm. Finally, we finish with some conclusions.

The main results of Sections 2 and 3 originally were sketched in our extended abstract [8], while the main result of Section 4 was published in Rus-

sian in [16]. Additionally to refined presentation of results [8, 16], the present paper contains a DP-based EA with a strengthened runtime bound for the case of DP algorithm with homogeneous transition functions (applicable e.g. to the shortest path problems).

2. Dynamic Programming

Dynamic programming is a general design paradigm for algorithms. The basic idea is to divide a problem into subproblems of the same type, and to construct a solution for the whole problem using the solutions for the subproblems. Dynamic programming has been proven to be effective for many single-objective as well as multi-objective optimization problems. It is even the most efficient approach known for solution of some problems in scheduling [35, 43], bioinformatics [6], routing (see e.g. [7], Chapters 24, 25) and other areas.

In this section, we will assume that an original optimization problem Π (single-objective or multi-objective) may be transformed into a multi-objective optimization problem P of a special type. The general scheme of dynamic programming will be presented and studied here in terms of the problem P. Several examples of a transformation from Π to P are provided at the end of the section.

2.1. Multi-Objective Optimization Problem

Let us consider a multi-objective optimization problem P which will be well suited for application of the DP algorithm in some sense, as shown below. Suppose, there are $d \in \mathbb{N}$ objectives that have to be optimized in P. An instance of problem P is defined by a quadruple $(d, g, \mathcal{S}, \mathcal{D})$. Here $g \colon \mathcal{S} \to (\mathbb{R} \cup \{\infty\})^d$ is called the *objective function*, $\mathcal{S}$ is called the *search space*, and $g(\mathcal{S}) \subseteq (\mathbb{R} \cup \{\infty\})^d$ is the *objective space*. $\mathcal{D} \subseteq \mathcal{S}$ is a set of feasible solutions.

We introduce the following partial order to define the goal in multi-objective optimization formally. Throughout this paper, $\preceq$ denotes *Pareto dominance* where

$$(y_1, \ldots, y_d) \preceq (y'_1, \ldots, y'_d)$$

iff $y_i \geq y'_i$ for all i for minimization criteria g_i and $y_i \leq y'_i$ for maximization criteria g_i. In the following, we use the notation $y' \prec y$ as an abbreviation for $y' \preceq y$ and $y \npreceq y'$. The *Pareto front* is the subset of $g(\mathcal{D})$ that consists

of all maximal elements of $g(\mathcal{D})$ with respect to $\preceq$. The goal is to determine a *Pareto-optimal set*, that is, a minimal by inclusion subset of feasible solutions $\mathcal{D}$ that is mapped on the Pareto front.

2.2. Framework for Dynamic Programs

Consider a DP algorithm for a problem P, working through a number of iterations called *phases*. In each phase the DP algorithm constructs and stores some *states* belonging to $\mathcal{S}$. By saying that DP algorithm computes a Pareto-optimal set for the problem P we mean that after completion of the DP algorithm, the set of all DP states produced at the final phase is a Pareto-optimal set for P.

Application of the DP approach to many multi-objective and single-objective optimization problems can be viewed as a transformation of a given problem Π to some problem P: a DP algorithm is applied to compute a Pareto-optimal set for P and this set is efficiently transformed into a solution to the given single- or multi-objective problem.

In what follows, we consider only those DP algorithms where the states of the current phase are computed by means of *transition functions*, each such function depending on the input parameters of problem P and taking as an argument some state produced at the previous phase.

Let us start the formal definition of the DP algorithm from a simplified version. Suppose that the simplified DP algorithm works in n phases, such that in the i-th phase a set $\mathcal{S}_i \subseteq \mathcal{S}$ of states is created. We use n finite sets $\mathcal{F}_i$ of state transition functions $F \colon \mathcal{S} \to \mathcal{S}'$ to describe the DP algorithm. Here $\mathcal{S}'$ is an extension of space $\mathcal{S}$. A mapping F can produce elements $F(S) \in \mathcal{S}' \backslash \mathcal{S}$ that do not belong to a search space. To discard such elements at phase i, $i = 1, \ldots, n$, a *consistency function* H_i is used, $H_i \colon \mathcal{S}' \to \mathbb{R}$, such that $S \in \mathcal{S}$ iff $H_i(S) \leq 0$. We assume that the number n, the functions H_i and the sets of functions $\mathcal{F}_i$ depend on the input instance of problem P.

The simplified DP algorithm proceeds as follows. In the initialization phase, the state space $\mathcal{S}_0$ is initialized with a finite subset of $\mathcal{S}$. In the i-th phase, the state space $\mathcal{S}_i$ is computed using the state space $\mathcal{S}_{i-1}$ according to

$$\mathcal{S}_i = \{F(S) \mid S \in \mathcal{S}_{i-1} \wedge F \in \mathcal{F}_i \wedge H_i(F(S)) \leq 0\}. \tag{1}$$

In the process, the consistency functions H_i serve to keep the infeasible elements emerging in phase i from being included into the current state space $\mathcal{S}_i$. (Note that after completion of phase n of the simplified DP algorithm, the

set $\mathcal{S}_n$ may contain some states whose objective values are Pareto-dominated by those of other states from $\mathcal{S}_n$.)

To delete the states with Pareto-dominated objective values and to improve the runtime of the simplified DP algorithm defined by (1), most of the practical DP algorithms utilize the Bellman principle (see e. g. [3]) or its variations so as to dismiss unpromising states without affecting the optimality of the final set of solutions. A formulation of the Bellman principle in terms of recurrence (1) for the single-objective problems can be found in Appendix A. Sufficient conditions for application of the Bellman principle in the single-objective case were formulated in [28]. In the multi-objective case the Bellman principle is not used, but the unpromising states may be excluded by means of an appropriate dominance relation on the set of states. Originally such dominance relations were introduced by R. Klötzler [25]. In this paper, we employ a similar approach, motivated by [43].

Let us consider a partial quasi-order (i. e. a reflexive and transitive relation) $\preceq_{\text{dom}}$ defined on $\mathcal{S}$ so that $S \preceq_{\text{dom}} S'$ iff $g(S) \preceq g(S')$. We will say that state S is *dominated* by state S' iff $S \preceq_{\text{dom}} S'$. If $S \in \mathcal{T} \subseteq \mathcal{S}$ is such that no $S' \in \mathcal{T}$ exists satisfying $S \preceq_{\text{dom}} S'$, then S will be called *non-dominated* in $\mathcal{T}$.

As we will see, under the following two conditions the relation $\preceq_{\text{dom}}$ is helpful to dismiss unpromising states in the DP algorithm.

The first condition C.1 guarantees that the dominance relation between two states transfers from one round to the next:

Condition C.1. For any $S, S' \in \mathcal{S}_{i-1}, i = 1, \ldots, n$, if $S \preceq_{\text{dom}} S'$ then $F(S) \preceq_{\text{dom}} F(S')$ for all $F \in \mathcal{F}_i$.

The second condition C.2 expresses that infeasible states cannot dominate feasible states:

Condition C.2. For any $S, S' \in \mathcal{S}'$, if $S \preceq_{\text{dom}} S'$ and $H_i(S) \leq 0$ then $H_i(S') \leq 0$.

Consider a subset $\mathcal{S}_i$ of $\mathcal{S}$. We call $\mathcal{T}_i \subseteq \mathcal{S}_i$ a *dominating subset* of $\mathcal{S}_i$ with respect to $\preceq_{\text{dom}}$ iff for any state $S \in \mathcal{S}_i$ there is a state $S' \in \mathcal{T}_i$ with $S \preceq_{\text{dom}} S'$. Let us use the notation $M(\mathcal{S}_i, \preceq_{\text{dom}})$ to denote the set of all dominating subsets of $\mathcal{S}_i$ which are minimal by inclusion.

The following proposition indicates that under conditions C.1 and C.2 it is sufficient to keep a dominating subset of states constructed in each phase i, rather than the full subset $\mathcal{S}_i$.

7

Proposition 1. *Suppose the simplified DP algorithm is defined by (1), conditions C.1 and C.2 hold and the dominating sets $\mathcal{T}_i$, $i = 1, \ldots, n$ are computed so that $\mathcal{T}_0 \in M(\mathcal{S}_0, \preceq_{\mathrm{dom}})$,*

$$\mathcal{T}_i \in M(\{F(S) \mid S \in \mathcal{T}_{i-1} \wedge F \in \mathcal{F}_i \wedge H_i(F(S)) \leq 0\}, \preceq_{\mathrm{dom}}). \qquad (2)$$

Then for any state $S^* \in \mathcal{S}_i$, $i = 0, \ldots, n$, there exists $S \in \mathcal{T}_i$ such that $S^* \preceq_{\mathrm{dom}} S$.

Proof. The proof is by induction on i. For $i = 0$ the statement holds by assumption $\mathcal{T}_0 \in M(\mathcal{S}_0, \preceq_{\mathrm{dom}})$.

By (1), a state $S^* \in \mathcal{S}_i$ can be expressed as $S^* = F^*(S')$, so that $H_i(S^*) \leq 0$, $F^* \in \mathcal{F}_i$ and $S' \in \mathcal{S}_{i-1}$. But by induction hypothesis, there exists a state $S^\diamond \in \mathcal{T}_{i-1}$ such that $S' \preceq_{\mathrm{dom}} S^\diamond$. Now conditions C.1 and C.2 imply that $S^* = F^*(S') \preceq_{\mathrm{dom}} F^*(S^\diamond)$ and $F^*(S^\diamond) \in \{F(S) \mid S \in \mathcal{T}_{i-1} \wedge F \in \mathcal{F}_i \wedge H_i(F(S)) \leq 0\}$. Hence, by (2) we conclude that there exists $S \in \mathcal{T}_i$ such that $S^* \preceq_{\mathrm{dom}} F^*(S^\diamond) \preceq_{\mathrm{dom}} S$. $\qquad\square$

In view of definition of $\preceq_{\mathrm{dom}}$, if the conditions of Proposition 1 are satisfied and the Pareto front of g is contained in $g(\mathcal{S}_n)$, then this Pareto front is also contained in $g(\mathcal{T}_n)$.

Proposition 2. *If the conditions of Proposition 1 are satisfied, then the size of each $\mathcal{T}_i$, $i = 0, \ldots, n$, is uniquely determined.*

Indeed, consider the set of maximal elements of $\mathcal{S}_i$ with respect to $\preceq_{\mathrm{dom}}$. Define the equivalence classes of this set with respect to the equivalence relation $x \equiv y$ iff $x \preceq_{\mathrm{dom}} y$ and $y \preceq_{\mathrm{dom}} x$. The size of a minimal subset M of the set of maximal elements, which dominates all elements of $\mathcal{S}_i$, is unique since such M contains one representative element from each equivalence class. $\qquad\square$

A computation satisfying (2) can be expressed in an algorithmic form as presented in Algorithm 1. It is easy to see that when a subset $\mathcal{T}_i$ is completed in Lines 8-13, condition (2) holds.

The runtime of a DP algorithm depends on the computation times for the state transition functions $F \in \mathcal{F}_i$, for the consistency functions H_i, for checking the dominance and manipulations with the sets of states. Let θ_F be an upper bound on computation time for a transition function F and let $\theta_{\mathcal{H}}$ be an upper bound for computation time of any function H_i, $i = 1, \ldots, n$.

Algorithm 1 Dynamic Program for P

1: $\mathcal{T}_0 \leftarrow \emptyset$
2: **for** $S \in \mathcal{S}_0$ **do**
3:    **if** $\nexists S' \in \mathcal{T}_0 \colon S \preceq_{\text{dom}} S'$ **then**
4:       $\mathcal{T}_0 \leftarrow (\mathcal{T}_0 \setminus \{S' \in \mathcal{T}_0 \mid S' \prec_{\text{dom}} S\}) \cup \{S\}$
5:    **end if**
6: **end for**
7: **for** $i = 1$ to n **do**
8:    $\mathcal{T}_i \leftarrow \emptyset$
9:    **for** $S \in \mathcal{T}_{i-1}$ and $F \in \mathcal{F}_i$ **do**
10:       **if** $H_i(F(S)) \leq 0$ and $\nexists S' \in \mathcal{T}_i \colon F(S) \preceq_{\text{dom}} S'$ **then**
11:         $\mathcal{T}_i \leftarrow (\mathcal{T}_i \setminus \{S' \in \mathcal{T}_i \mid S' \prec_{\text{dom}} F(S)\}) \cup \{F(S)\}$
12:       **end if**
13:    **end for**
14: **end for**
15: **return** $\mathcal{T}_n$

Sometimes it will be appropriate to use the average computation time for the state transition functions at phase i, $i = 1, \ldots, n$: $\theta_{\mathcal{F}_i} = \sum_{F \in \mathcal{F}_i} \theta_F / |\mathcal{F}_i|$.

In Algorithm 1, verification of condition

$$\nexists S' \in \mathcal{T}_i \colon F(S) \preceq_{\text{dom}} S' \tag{3}$$

in Line 10 and execution of Line 11 may be implemented using similar problem-specific data structures. To take this into account, we will denote by $\theta_{\preceq}$ an upper bound applicable both for the time to verify (3) and for the time to execute Line 11.

The body (lines 10–12) of the main loop (Lines 7–14) in Algorithm 1 is executed $\sum_{i=1}^{n} |\mathcal{T}_{i-1}| \cdot |\mathcal{F}_i|$ times.

To simplify the subsequent analysis let us assume that in the if-statement at line 10, the condition (3) is always checked. We denote the computation time for initializing $\mathcal{T}_0$ with θ_{ini} (Lines 1–6) and the computation time for presenting the result with θ_{out} (Line 15), which leads to an overall runtime

$$O\left(\theta_{\text{ini}} + \sum_{i=1}^{n} |\mathcal{F}_i| \cdot |\mathcal{T}_{i-1}| \cdot (\theta_{\mathcal{F}_i} + \theta_{\mathcal{H}} + \theta_{\preceq}) + \theta_{\text{out}}\right). \tag{4}$$

In many applications of the DP, the computation time for the state transition functions and the consistency functions are constant. Besides that,

the partial quasi-order $\succeq_{\text{dom}}$ is often just a product of linear orders and it is sufficient to allocate one element in a memory array to store one (best found) element for each of the linear orders. This data structure usually allows to verify (3) and to execute Line 11 in constant time (see the examples in Subsection 3.5). In the cases mentioned above, the values θ_F, $\theta_{\mathcal{H}}$ and $\theta_{\preceq}$ can be chosen equal to the corresponding computation times and the overall DP algorithm runtime in (4) can be expressed with symbol $\Theta(\cdot)$ instead of $O(\cdot)$.

Note that the runtime of the DP algorithm is polynomially bounded in the input length of problem P if θ_{ini}, n, $\theta_{\mathcal{F}_i}$, $\theta_{\mathcal{H}}$, $\theta_{\preceq}$, θ_{out}, as well as $|\mathcal{T}_i|$ and $|\mathcal{F}_{i+1}|$ for $i = 0, \ldots, n-1$, are polynomially bounded in the input length. Here and below, we say that a value (e.g. the running time) is polynomially bounded in the input length, meaning that there exists a polynomial function of the input length, which bounds the value from above.

2.3. Applications of the general DP scheme

In this subsection, we point out how the general DP framework presented above is applied to some classical combinatorial optimization problems. The approach followed here is to describe the appropriate problem P and the components of a dynamic programming algorithm for the solution of a specific problem Π. Most of the following examples have been inspired by the previous works [9, 39, 41]. Note that $\preceq_{\text{dom}}$ will be a product of linear orders in each of these examples. In what follows id denotes the identical mapping.

Traveling Salesman Problem. Let us first consider the traveling salesman problem (TSP) as a prominent NP-hard example. The input for the TSP consists of a complete graph $\mathcal{G} = (V, E)$ with a set of nodes $V = \{1, 2, \ldots, n\}$ and non-negative edge weights $w \colon E \to \mathbb{R}_0^+$. It is required to find a permutation of all nodes $(v_1, \ldots, v_n)$, such that the TSP tour length $\sum_{i=2}^n w(v_{i-1}, v_i) + w(v_n, v_1)$ is minimized. Without loss of generality we can assume that $v_1 = 1$, that is, the TSP tour starts in the fixed vertex 1.

The search space $\mathcal{S}$ for problem P corresponding to the dynamic programming algorithm of Held and Karp [21] consists of all paths $S = (v_1, \ldots, v_i), v_1 = 1$ of $i = 1, \ldots, n$ nodes. $\mathcal{S}'$ is the extended search space of all sequences of nodes up to length n (the same node may occur more than once). Given $M \subseteq V \backslash \{1\}$ and $k \in M$, let $\pi(k, M)$ denote the set of all paths of $|M| + 1$ vertices starting in vertex 1 then running over all nodes from M and ending in vertex k. Let the vector objective function $g : \mathcal{S} \to (\mathbb{R} \cup \{\infty\})^d$, $d = (n-1)2^{(n-1)}$ have components $g_{kM}(S)$ for all $M \subseteq V \backslash \{1\}$, $k \in M$, equal

to the length of path S iff $S \in \pi(k, M)$. For all other $S \notin \pi(k, M)$ assume $g_{kM}(S) = \infty$. The set of feasible solutions is $\mathcal{D} = \cup_{k=2}^n \pi(k, V \setminus \{1\})$, since in the TSP we seek a tour of length n.

$\mathcal{S}_0$ consists of a single element v_1. The set $\mathcal{F}_i$ for all i consists of $n - 1$ functions $F_v : \mathcal{S} \to \mathcal{S}'$ that add vertex $v \in V \setminus \{1\}$ to the end of the given path. For invalid states $S \in \mathcal{S}'$, which are characterized by not being Hamiltonian paths on their vertex sets, the mapping $H_i(S)$ computes 1 and 0 otherwise.

In view of the definition of objective g, the dominance relation is formulated as follows. $S \preceq_{\text{dom}} S'$ if and only if S and S' are Hamiltonian paths on the same ground set with the same end vertex k and path S' is not longer than S. States from different sets $\pi(k, M)$ are not comparable. Conditions C.1 and C.2 are verified straightforwardly.

Substituting these components into Algorithm 1, we get almost the whole well-known dynamic programming algorithm of Held and Karp [21], except for the last step where the optimal tour is constructed from the optimal Hamiltonian paths.

Algorithm 1 initializes the states of the dynamic program with paths $(1, v)$ for all $v \in V \setminus \{1\}$. In each subsequent iteration i, the algorithm takes each partial solution S obtained in the preceding iteration and checks for every application of the state transition function $F(S)$ with $F \in \mathcal{F}_i$ whether $H_i(F(S))$ is a feasible partial solution that is non-dominated in $\mathcal{T}_i$. If so, then $F(S)$ is added to the set $\mathcal{T}_i$ of new partial solutions by replacing dominated partial solutions S' defined on the same ground set with the same end vertex of the Hamiltonian path.

What remains to do after completion of the DP algorithm with Pareto-optimal set is to output the Pareto-optimal solution minimizing the criterion $g_{k, V \setminus \{1\}}(S) + w(k, 1), k \in V \setminus \{1\}$, which is now easy to find. Here using appropriate data structures one gets $\theta_F = \Theta(1)$, $\theta_{\mathcal{H}} = \Theta(1)$, $\theta_{\preceq} = \Theta(1)$ and $|\mathcal{T}_i| = i \binom{n-1}{i}$, $|\mathcal{F}_i| = n - 1$ for all $i = 1, \ldots, n$, thus the observation following (4) leads to the time complexity bound $\Theta(n^2 2^n)$.

Knapsack Problem. Another well-known NP-hard combinatorial optimization problem that can be solved by dynamic programming is the knapsack problem. The input for the knapsack problem consists of n items where each item i has an associated integer weight $w_i > 0$ and profit $p_i > 0$, $1 \leq i \leq n$. Additionally a weight bound W is given. The goal is to determine an item selection $K \subseteq \{1, \ldots, n\}$ that maximizes the profit $\sum_{i \in K} p_i$, subject to the condition $\sum_{i \in K} w_i \leq W$.

We fit the problem into the above framework assuming that each state $S = (s_1, s_2) \in \mathcal{S}_i$, $i = 1, \ldots, n$, encodes a partial solution for the first i items, where coordinate s_1 stands for the weight of a partial solution and s_2 is its profit. The initial set $\mathcal{S}_0$ consists of a single element $(0,0)$ encoding a selection of no items.

The pseudo-Boolean vector function $g \colon \mathcal{S} \to \mathbb{R}^W$ defines W criteria

$$g_w(S) := \begin{cases} s_2 & \text{if } s_1 = w \\ 0 & \text{otherwise} \end{cases} , \quad w = 0, \ldots, W, \tag{5}$$

that have to be maximized. This implies the dominance relation $\preceq_{dom}$ such that $S \preceq_{\text{dom}} S'$ iff $s_1 = s_1'$ and $s_2 \leq s_2'$, where $S = (s_1, s_2)$, $S' = (s_1', s_2')$.

The set $\mathcal{F}_i$ consists of two functions: id and $F_i(s_1, s_2) = (s_1 + w_i, s_2 + p_i)$. Here F_i corresponds to adding the i-th item to the partial solution, and id corresponds to skipping this item. A new state $S = (s_1, s_2)$ is accepted if it does not violate the weight limit, i.e. $H_i(S) \leq 0$, where $H_i(S) = s_1 - W$.

The conditions C.1 and C.2 are straightforwardly verified. To obtain an optimal solution for the knapsack problem it suffices to select the Pareto-optimal state with a maximal component s_2 from $\mathcal{S}_n$.

To reduce the comparison time $\theta_\preceq$ we can store the states of the DP in a $(W \times n)$-matrix. An element in row w, $w = 1, \ldots, W$, and column i, $i = 1, \ldots, n$, holds the best value s_2 obtained so far on states $S = (s_1, s_2) \in \mathcal{T}_i$ with $s_1 = w$. Then $\theta_\preceq$ is a constant and the worst-case runtime of the explained DP algorithm is $O(n \cdot W)$ since $\sum_{i=1}^n |\mathcal{T}_{i-1}| \leq nW$.

Single Source Shortest Path Problem. A classical problem that also fits into the DP framework is the single source shortest path problem (SSSP). Given an undirected connected graph $\mathcal{G} = (V, E)$, $|V| = n$ and positive edge weights $w \colon E \to \mathbb{R}^+$, the task is to find shortest paths from a selected *source* vertex $s \in V$ to all other vertices.

The search space $\mathcal{S}$ is a set of all paths in $\mathcal{G}$ with an end-point s. The set of feasible solutions $\mathcal{D}$ is equal to $\mathcal{S}$.

Since adding a vertex to a path may result in a sequence of vertices that do not constitute a path in $\mathcal{G}$, we extend the search space to the set $\mathcal{S}'$ of all sequences of vertices of length at most n with an end-point s. The set $\mathcal{S}_0$ of initial solutions is just a single vertex s. Now for all i, we define $\mathcal{F}_i := \{F_v \mid v \in V\} \cup \{\text{id}\}$, where $F_v \colon \mathcal{S} \to \mathcal{S}'$ is the mapping adding the vertex v to a sequence of vertices. $H_i(S) = -1$ if S is a path in $\mathcal{G}$ with an end-point s, and 1 if not.

Let the vector objective function g have $d = n$ components $g_v(S)$ for all $v \in V$, equal to the length of path S iff S connects s to v, otherwise assume $g_v(S) = \infty$. This implies that $S \preceq_{\mathrm{dom}} S'$ if and only if the paths S and S' connect s to the same vertex and S' is not longer than S.

The resulting DP algorithm has $\theta_F = \Theta(1)$, $\theta_{\mathcal{H}} = \Theta(1)$, $\theta_{\preceq} = \Theta(1)$ and $|\mathcal{T}_{i-1}| = \Theta(n)$, $|\mathcal{F}_i| = \Theta(n)$ for all $i = 1, \ldots, n$, thus (4) gives the time complexity bound $O(n^3)$. The well-known Dijkstra's algorithm has $O(n^2)$ time bound, but in that algorithm only one transition mapping is applied in each phase (attaching the closest vertex to the set of already reached ones), and such a problem-specific DP scheme is not considered here.

All-Pairs Shortest Path Problem. Finally, let us consider the all-pairs shortest path (APSP) problem, which has the same input as the SSSP, except that no source vertex is given, and the goal is to find for each pair (u, v) of vertices a shortest path connecting them.

A basic observation is that sub-paths of shortest paths are shortest paths again. Hence a shortest path connecting u and v can be obtained from appending the edge (x, v), where x is a neighbor of v, to a shortest path from u to x. This allows a very natural DP formulation as described for problem P.

For the APSP, the search space $\mathcal{S}$ naturally is the set of all paths in $\mathcal{G}$, and the set $\mathcal{D}$ of feasible solutions consists of collections of paths, where for each pair of vertices there is one path connecting them.

We model paths via finite sequences of vertices, and do not allow cycles. Since adding a vertex to a path may create a sequence of vertices which does not correspond to a path in $\mathcal{G}$, let us extend this search space to the set $\mathcal{S}'$ of all sequences of vertices of length at most n. The set $\mathcal{S}_0$ of initial solutions is the set of all paths of length 0, that is, of all sequences consisting of a single vertex. Now for all i, we define $\mathcal{F}_i := \{F_v \mid v \in V\} \cup \{\mathrm{id}\}$, where $F_v : \mathcal{S}' \to \mathcal{S}'$ is the mapping adding the vertex v to a sequence of vertices. To exclude invalid solutions, let us define $H_i(S)$ to be -1 if S is a path in $\mathcal{G}$, and 1 if not.

It remains to define when one state dominates another. Let π_{ij} denote the set of all paths starting in vertex i and ending in vertex j. Let the vector objective function $g : \mathcal{S} \to (\mathbb{R} \cup \{\infty\})^d$, $d = n^2$ have components $g_{ij}(S)$ for all $i, j \in V$, equal to the length of path S iff $S \in \pi_{ij}$. For all other $S \notin \pi_{ij}$ assume $g_{ij}(S) = \infty$. This implies that $S \preceq_{\mathrm{dom}} S'$ if and only if the paths S and S' connect the same two vertices and S' is not longer than S.

13

Since the length of the path arising from extending an existing path by an edge depends monotonically on the length of the existing path, conditions C.1 and C.2 hold. So, in view of Proposition 1, any set $\mathcal{T}_i$ contains a path for each pair of vertices (and only one such path). Thus, $\mathcal{T}_n$ is a subset of $\mathcal{D}$ and contains a shortest path for any pair of vertices.

The resulting algorithm following the dynamic programming approach now does the following. It starts with all paths of length zero as solution set $\mathcal{S}_0$. It then repeats n times the following. For each path in the solution set and each vertex, it appends the vertex to the path. If the resulting path dominates an existing solution with the same end vertices, it replaces the latter. Here $\theta_F = \Theta(1)$, $\theta_{\mathcal{H}} = \Theta(1)$, $\theta_{\preceq} = \Theta(1)$ and $|\mathcal{T}_i| = O(n^2)$, $|\mathcal{F}_i| = O(n)$ for all $i = 1, \ldots, n$, thus (4) gives the time complexity bound $O(n^4)$. Note that the well-known Floyd-Warshall algorithm (see e.g. [7], Chapter 25) has $O(n^3)$ time bound, but in that algorithm each transition mapping combines two states (paths), and such an option is not considered in this paper.

3. Evolutionary Algorithms

In the following, we show how results of dynamic programming can be attained by evolutionary algorithms. To this aim, we state a general formulation of such an evolutionary algorithm and then describe how the different components have to be designed.

3.1. Framework for Evolutionary Algorithms

An evolutionary algorithm consists of different generic modules, which have to be made precise by the user to best fit to the problem. Experimental practice, but also some theoretical work (see e.g. [10, 11, 12, 29]), demonstrate that the right choice of representation, variation operators, and selection method is crucial for the success of such algorithms.

We assume again that an instance of problem P is given by a multi-objective function g that has to be optimized. We consider simple evolutionary algorithms that consist of the following components.

We use $\mathcal{S}'_{\mathrm{EA}} := \{0, \ldots, n\} \times \mathcal{S}'$ as the *phenotype space* and call its elements *individuals*. The algorithm (see Algorithm 2) starts with an initial population of individuals $\mathcal{P}_0$. During the optimization the evolutionary algorithm uses a selection operator $\mathrm{sel}(\cdot)$ and a mutation operator $\mathrm{mut}(\cdot)$ to create new individuals. The d-dimensional objective function together with a partial order $\preceq$ on $\mathbb{R}^d$ induce a partial quasi-order $\preceq_{\mathrm{EA}}$ on the phenotype space,

Algorithm 2 Evolutionary Algorithm for P

1: $\mathcal{P} \leftarrow \emptyset$
2: **for** $I \in \mathcal{P}_0$ **do**
3:    **if** $\nexists I' \in \mathcal{P} : I \prec_{\text{EA}} I'$ **then**
4:       $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{I' \in \mathcal{P} \mid I' \prec_{\text{EA}} I\}) \cup \{I\}$
5:    **end if**
6: **end for**
7: **loop**
8:    $I \leftarrow \text{mut}(\text{sel}(\mathcal{P}))$
9:    **if** $\nexists I' \in \mathcal{P} : I \prec_{\text{EA}} I'$ **then**
10:       $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{I' \in \mathcal{P} \mid I' \prec_{\text{EA}} I\}) \cup \{I\}$
11:    **end if**
12: **end loop**
13: **return** $\{\text{out}_{\text{EA}}(I) \mid I = (i, S) \in \mathcal{P}, S \in \mathcal{D}\}$

---

which guides the search. After the termination of the EA, an output function $\text{out}_{\text{EA}}(\cdot)$ is utilized to map the individuals in the last population to search points from the DP search space.

3.2. Defining the Modules

We now consider how the different modules of the evolutionary algorithm have to be implemented so that it can carry out dynamic programming. To do this, we relate the modules to the different components of a DP algorithm. Consider a problem P given by a set of feasible solutions $\mathcal{D}$ and a multi-objective function g that can be solved by a dynamic programming approach. The EA works with the following setting.

The initial population is $\mathcal{P}_0 = \{0\} \times \mathcal{S}_0$ where $\mathcal{S}_0$ is the initial state space of the DP algorithm. The selection operator $\text{sel}(\cdot)$ chooses an individual $I \in \mathcal{P}$ the following way. First it chooses $i \leq n-1$ uniformly from the set of phases which are represented in the current population i.e. from the set $\{k : k \leq n-1, \exists (k, S) \in \mathcal{P}\}$. After this, selection chooses I uniformly among the individuals of the form (i, S) in the current population.

For an individual (i, S), the mutation operator $\text{mut}(\cdot)$ chooses a state transition function $F \in \mathcal{F}_{i+1}$ uniformly at random and sets $\text{mut}((i, S)) = (i+1, F(S))$.

We incorporate a partial order $\preceq_{\text{EA}}$ into the EA to guide the search. This

relation is defined as follows:

$$(i, S) \preceq_{\mathrm{EA}} (i', S') \Leftrightarrow (i = i' \text{ and } S \preceq_{\mathrm{dom}} S') \text{ or } H_i(S) > 0. \qquad (6)$$

Finally, we utilize the output function $\mathrm{out}_{\mathrm{EA}}((i, S)) = S$ to remove the additional information at the end of a run of the EA. That is, we remove the information that was used to store the number of a certain round of the underlying dynamic program and transform an individual into a search point for the problem P.

Note that the description of the Algorithm 2 does not employ the notion of the fitness function, although an appropriate multi-objective fitness function may be defined for compatibility with the standard EA terminology.

Finally, note that we do not discuss the solutions encoding in our EA because it is not essential for the analysis. However, it may be worth mentioning, when the biological analogy is considered. Here each of the genes A_i, $i = 1, \ldots, n$ would define the DP transition mapping from a set $\mathcal{F}_i$, and a composition of these mappings would yield the DP state represented by the individual. One of the possible options of each gene is "undefined", and the mutation operator modifies the first gene which is still "undefined" in the parent individual. A discussion of genetic mechanisms corresponding to the proposed mutation in a biological system is provided in Appendix B.

3.3. Runtime of the Evolutionary Algorithm

Our goal is to show that the evolutionary algorithm solves the problem P efficiently if the dynamic programming approach does. To measure the time the evolutionary algorithm needs to compute a Pareto-optimal set for problem P, one would analyze the expected number of fitness evaluations to come up with a Pareto-optimal set, when it is non-empty. This is also called the expected *optimization time*, which is a common measure for analyzing the runtime behavior of evolutionary algorithms. The proposed EA does not use the multi-objective fitness function explicitly, but given enough memory, it may be implemented so that every individual constructed and evaluated in Lines 3 and 4 or in Lines 8-11 requires at most one evaluation of the objective function g. Thus, we can define the optimization time for Algorithm 2 as $|\mathcal{S}_0|$ plus the number of iterations of the main loop (Lines 8-11) required to come up with a Pareto-optimal set. Analogous parameter of a DP algorithm is the number of states computed during its execution.

The next theorem relates the expected optimization time of the EA to the number of states computed during the execution of the corresponding

DP algorithm. In what follows it will be convenient to denote the cardinality of the set of states produced after completion of the DP algorithm by $|DP|$, i. e. $|DP| := \sum_{i=0}^{n} |\mathcal{T}_i|$. Note that $|DP|$ is a well-defined value since the sizes $|\mathcal{T}_i|$ are unique according to Proposition 2.

Theorem 1. *Let a DP be defined as in Algorithm 1 and an EA defined as in Algorithm 2 with $\preceq_{\mathrm{EA}}$ relation defined by (6). Then the number of states computed during the execution of the DP algorithm is $|\mathcal{S}_0| + \sum_{i=1}^{n} |\mathcal{F}_i| \cdot |\mathcal{T}_{i-1}|$, and the EA has an expected optimization time of*

$$
O\left(|\mathcal{S}_0| + n \cdot \log |DP| \cdot \sum_{i=0}^{n-1} |\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}| \right).
$$

Proof. Estimation of the number of states computed during the execution of the DP algorithm is straightforward.

Assume that the optimization process works in stages $1 \leq i \leq n$, whereas stage $i+1$ starts after the stage i has been finished. We define that a stage i finishes when for every state $S \in \mathcal{T}_i$ there exists an individual $(i, S') \in \mathcal{P}$ with S' dominating S. Here and below in this proof, by $\mathcal{T}_i$, $i = 0, \ldots, n$, we denote the corresponding sets computed in Algorithm 1. Note that after completion of a stage i, the subset of individuals of a form (i, S) in population $\mathcal{P}$ does not change in the subsequent iterations of the EA. Let $\mathcal{T}_i'$ denote the set of states of these individuals after completion of stage i, $i = 0, \ldots, n$. By the definition of Algorithm 2, the sequence $\mathcal{T}_i'$, $i = 0, \ldots, n$ satisfies (2), and therefore $|\mathcal{T}_i'| = |\mathcal{T}_i|$, $i = 0, \ldots, n$ in view of Proposition 2.

Let ξ_i be the random variable denoting the number of iterations since stage $i-1$ is finished, until stage i is completed. Then the expected optimization time is given by $|\mathcal{S}_0| + E[\xi]$ with $\xi = \xi_1 + \ldots + \xi_n$.

Any state $S \in \mathcal{T}_{i+1}$ is computed in Algorithm 1 by means of some function $\tilde{F} \in \mathcal{F}_{i+1}$, when it is applied to some state $\tilde{S} \in \mathcal{T}_i$. Thus, in stage $i+1$ of the EA during mutation the same transition function $\tilde{F}$ may be applied to some individual $I' = (i, S')$, such that $\tilde{S} \preceq_{\mathrm{dom}} S'$. After this mutation, in view of conditions C.1 and C.2, the population $\mathcal{P}$ will contain an individual $I'' = (i+1, S'')$ with S'' such that $S \preceq_{\mathrm{dom}} \tilde{F}(S') \preceq_{\mathrm{dom}} S''$.

Consider any iteration of the EA at stage $i+1$. Let t denote the number of such states from $\mathcal{T}_{i+1}$ that are already dominated by a state of some individual in $\mathcal{P}$. Then there should be $|\mathcal{T}_{i+1}| - t$ new individuals of the form $(i+1, S)$ to be added into $\mathcal{P}$ to complete stage $i+1$ (recall that $|\mathcal{T}'_{i+1}| = |\mathcal{T}_{i+1}|$). The

17

probability to produce an individual (i, S') where S' dominates a previously non-dominated state from $\mathcal{T}_{i+1}$ is no less than $(|\mathcal{T}_{i+1}| - t)/(n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|)$ with an expected waiting time of at most $(n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|)/(|\mathcal{T}_{i+1}| - t)$ for this geometrically distributed variable. The expected waiting time to finish stage $i + 1$ is thus bounded by

$$E[\xi_{i+1}] \leq \sum_{t=1}^{|\mathcal{T}_{i+1}|} \frac{n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|}{t} = n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}| \cdot \mathcal{H}_{|\mathcal{T}_{i+1}|},$$

with $\mathcal{H}_k$ being the k-th harmonic number, $\mathcal{H}_k := \sum_{i=1}^{k} \frac{1}{i}$.

This leads to an overall expected number of iterations

$$E[\xi] \leq \sum_{i=0}^{n-1} n|\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}| \cdot \mathcal{H}_{|\mathcal{T}_{i+1}|} \leq n(\ln |DP| + 1) \cdot \sum_{i=0}^{n-1} |\mathcal{T}_i| \cdot |\mathcal{F}_{i+1}|.$$

$\square$

A similar inspection as in Subsection 2.2 reveals that the expected runtime of the EA is

$$O\left(\theta_{\mathrm{ini}} + n \log |DP| \cdot \sum_{i=0}^{n-1} (|\mathcal{F}_{i+1}| \cdot |\mathcal{T}_i| \cdot (\theta_{\mathcal{F}_i} + \theta_{\mathcal{H}} + \theta_{\preceq})) + \theta_{\mathrm{out}}\right),$$

assuming the individuals of the population are stored in $n + 1$ disjoint sets according to the first coordinate i.

As noted in Subsection 2.2, if the computation times for functions F, H_i and dominance checking (3) as well as execution time for Line 11 in Algorithm 1 are constant, then θ_F, $\theta_{\mathcal{H}}$ and $\theta_{\preceq}$ can be chosen *equal* to the corresponding computation times. In such cases a problem that is solved by dynamic programming Algorithm 1 in time T, will be solved by the EA defined as in Algorithm 2 in expected time $O(Tn \log |DP|)$.

3.4. Homogeneous transitions

Some DP algorithms, like the ones for the APSP and SSSP problems, have a specific structure which may be exploited in the EA. In this subsection we consider the case of *homogeneous* transition functions where $\mathcal{F}_1 \equiv \ldots \equiv \mathcal{F}_n$ and $H_1 \equiv \ldots \equiv H_n$. To simplify the notation in this case we will assume

18

$\mathcal{F}_1 \equiv \mathcal{F}$ and $H_1(S) \equiv H(S)$. Additionally, we suppose that the identical mapping belongs to $\mathcal{F}$.

The formulated assumptions imply that once some state S is obtained in the DP algorithm, it will be copied from one phase to another, unless some other state will dominate it. Note also that it does not matter at what particular phase a state has been obtained – the transition functions will produce the same images of this state. These observations motivate a modification of the partial order $\preceq_{\text{EA}}$, neglecting the phase number in comparison of individuals:

$$(i, S) \preceq_{\text{EA}} (i', S') \Leftrightarrow S \preceq_{\text{dom}} S' \ \text{ or } H_i(S) > 0. \tag{7}$$

In fact, now we can skip the index i in individuals (i, S) of the EA, so in this subsection the terms "state" and "individual" are synonyms and the phase number i is suppressed in the notation of individuals. As the following theorem shows, wider sets of comparable individuals in this special case allow to reduce the population size and thus improve the performance of the EA. Let us consider the *width* W_{dom} of partial order $\preceq_{\text{dom}}$, i. e. the maximum size of a set of pairwise incomparable elements.

Theorem 2. *If the transition functions are homogeneous and $id \in \mathcal{F}$, then the EA defined as in Algorithm 2 with the modified $\preceq_{\text{EA}}$ relation (7) has an expected optimization time of $O(|\mathcal{S}_0| + W_{\text{dom}} \log(W_{\text{dom}}) \cdot n|\mathcal{F}|)$.*

Proof. The analysis is similar to the proof of Theorem 1. Note that now the size of population $\mathcal{P}$ does not exceed W_{dom}. We assume that $|\mathcal{P}| = W_{\text{dom}}$ right from the start.

Let $\mathcal{T}_i$ be the same as in phase i of the DP algorithm, $i = 0, \ldots, n$. Suppose again that the optimization process works in stages $1 \le i \le n$, whereas stage i is assumed to be finished when for every $S \in \mathcal{T}_i$, the population $\mathcal{P}$ contains an individual S' such that $S \preceq_{\text{dom}} S'$.

Let ξ_i be the number of iterations since stage $i-1$ is finished, until stage i is completed. Then the expected optimization time is given by $|\mathcal{S}_0| + E[\xi]$ with $\xi = \xi_1 + \ldots + \xi_n$.

Any state $S \in \mathcal{T}_{i+1}$ is computed in the DP algorithm by means of some function $\tilde{F} \in \mathcal{F}_{i+1}$, when it is applied to some state $\tilde{S} \in \mathcal{T}_i$. Thus, in stage $i+1$ of the EA during mutation the same transition function $\tilde{F}$ may be applied to some individual $I' = S'$, such that $\tilde{S} \preceq_{\text{dom}} S'$. After this mutation, in

view of conditions C.1 and C.2, the population $\mathcal{P}$ will contain an individual $I'' = S''$ such that $S \preceq_{\mathrm{dom}} \tilde{F}(S') \preceq_{\mathrm{dom}} S''$.

The probability of such a mutation for a particular $S \in \mathcal{T}_{i+1}$ is at least $1/(|\mathcal{F}_{i+1}| \cdot |\mathcal{P}|) \leq 1/(|\mathcal{F}_{i+1}| \cdot W_{\mathrm{dom}})$. Let t denote the number of states $S \in \mathcal{T}_{i+1}$ that are already dominated at stage $i+1$. Then there are at least $|\mathcal{T}_{i+1}| - t$ possibilities to add a new individual, which dominates a previously non-dominated state from $\mathcal{T}_{i+1}$. The probability for such a mutation is not less than $(|\mathcal{T}_{i+1}| - t)/(W_{\mathrm{dom}} \cdot |\mathcal{F}_{i+1}|)$ with an expected waiting time of at most $(W_{\mathrm{dom}} \cdot |\mathcal{F}_{i+1}|)/(|\mathcal{T}_{i+1}| - t)$ for this geometrically distributed variable. The expected waiting time to finish stage $i+1$ is thus $E[\xi_{i+1}] \leq W_{\mathrm{dom}} \cdot |\mathcal{F}_{i+1}| \cdot \mathcal{H}_{|\mathcal{T}_{i+1}|}$. But $|\mathcal{T}_{i+1}| \leq W_{\mathrm{dom}}$ because the states of $\mathcal{T}_{i+1}$ are pairwise incomparable according to Algorithm 1. This leads to an overall expected number of iterations $E[\xi] \leq W_{\mathrm{dom}} \cdot (\ln(W_{\mathrm{dom}}) + 1) \cdot \sum_{i=0}^{n-1} |\mathcal{F}_{i+1}|$. $\qquad\square$

3.5. Examples

Now, we point out how the framework presented in this section can be used to construct evolutionary algorithms using the examples from Section 2.

Traveling Salesman Problem. Due to Theorem 1 the expected optimization time of the evolutionary algorithm based on the DP algorithm of Held and Karp presented in Section 2.3 is $O(n^4 2^n)$. This bound can be further improved to $O(n^3 \cdot 2^n)$ for the EA proposed in [41].

Knapsack Problem. Consider the DP algorithm presented in Section 2.3. The expected optimization time of the corresponding EA for the knapsack problem is $O(n^2 \cdot W \cdot \log(n \cdot W))$ due to Theorem 1.

Single Source Shortest Path Problem. Application of Theorem 1 to the DP algorithm for SSSP problem from Section 2.3 gives an expected optimization time of $O(n^4 \log(n))$ for Algorithm 2.

The DP algorithm for SSSP problem has homogeneous transition functions with with $W_{\mathrm{dom}} = n$. Thus, the modified EA considered in Theorem 2 has the expected optimization time $O(n^3 \log n)$. This bound can be further improved to $O(n^3)$ for the (1+1)-EA [39].

All-Pairs Shortest Path Problem. Plugging the ideas of the DP algorithm for APSP problem presented in Section 2.3 into the framework of Algorithm 2,

we obtain an EA with an expected optimization time of $O(n^5 \log(n))$ due to Theorem 1.

It has been noted, however, that the DP algorithm for APSP has homogeneous transition functions, each set $\mathcal{F}_i$ contains the identical mapping. Here $W_{\mathrm{dom}} = n^2$, thus Theorem 2 implies that the modified EA has the expected optimization time $O(n^4 \log n)$. This algorithm can be further improved to an EA with optimization time $\Theta(n^4)$ as has been shown in [9].

4. Approximation Schemes

In this section, we demonstrate that for many single-objective discrete optimization problems Π the above framework can be used to find feasible solutions with any desired precision. The supplementary multi-objective problem P will be formally introduced for compatibility with the previous sections, but it will not play a significant role here.

Throughout this section we assume that Π is an NP-optimization problem [2], $\mathbf{x}$ denotes the input data of an instance of Π, $Sol_{\mathbf{x}}$ is the set of feasible solutions, $m_{\mathbf{x}} : Sol_{\mathbf{x}} \to \mathbb{N}_0$ is the objective function (here and below $\mathbb{N}_0$ denotes the set of non-negative integers). The optimal value of the objective function is $OPT(\mathbf{x}) = \max_{y \in Sol_{\mathbf{x}}} m_{\mathbf{x}}(y)$ if Π is a maximization problem, or $OPT(\mathbf{x}) = \min_{y \in Sol_{\mathbf{x}}} m_{\mathbf{x}}(y)$ in the case of minimization. To simplify presentation in what follows we assume that $Sol_{\mathbf{x}} \neq \emptyset$.

To formulate the main result of this section let us start with two standard definitions [18].

A *ρ-approximation algorithm* for Π is an algorithm that for any instance $\mathbf{x}$ returns a feasible solution whose objective value at most ρ times deviates from $OPT(\mathbf{x})$ (if the instance $\mathbf{x}$ is solvable). Such a solution is called *ρ-approximate*. A *fully polynomial time approximation scheme (FPTAS)* for a problem Π is a family of $(1 + \varepsilon)$-approximation algorithms over all factors $\varepsilon > 0$ with polynomially bounded running time in problem input size $|\mathbf{x}|$ and in $1/\varepsilon$.

In [43] G. Woeginger proposed a very general FPTAS with an outline similar to the DP Algorithm 1, except that the comparison of newly generated states to the former ones is modified so that the "close" states are not kept. This modified algorithm is denoted by DP_{Δ} in what follows (a detailed description of DP_{Δ} will be given in Subsection 4.1).

The state space $\mathcal{S}$ and its subsets $\mathcal{T}_i$ computed in the DP Algorithm 1 may be exponential in problem input size, thus leading to an exponential running

time of the DP algorithm (this holds e.g. for the Knapsack problem). The algorithm DP_Δ, however, iteratively thins out the state space of the dynamic program and substitutes the states that are "close" to each other by a single representative, thus bringing the size of the subsets $\mathcal{T}_i$ down to polynomial. This transformation is known as *trimming the state space* approach.

In [43], a list of conditions is presented, that guarantee the existence of an FPTAS when there is an exact DP algorithm for a problem. If a problem Π satisfies these conditions, it is called *DP-benevolent*. This class, in particular, contains the knapsack problem and different scheduling problems, e.g. minimizing the total weighted job completion time on a constant number of parallel machines, minimizing weighted earliness-tardiness about a common non-restrictive due date on a single machine, minimizing the weighted number of tardy jobs etc. The definition of DP-benevolence is as follows.

The input data of Π has to be structured so that $\mathbf{x}$ consists of n vectors $X_1, \ldots, X_n \in \mathbb{N}_0^\alpha$ and the components $x_{1i}, \ldots, x_{\alpha i}$ of each vector X_i are given in binary coding. The dimension α may depend on the specific problem input.

Suppose that for a problem Π there exists a corresponding multi-objective problem P and an exact simplified DP algorithm defined by expression (1). This algorithm works in n phases and for each $i = 1, \ldots, n$ the set of functions $\mathcal{F}_i$ and the function H_i do not depend on any input vectors other than X_i. Besides that, $\mathcal{S} \subset \mathcal{S}' = \mathbb{N}_0^\beta$, where dimension β is fixed for Π and does not depend on a particular input $\mathbf{x}$. The assumption that elements of $\mathcal{S}'$ are integer vectors will be essential in this section because each component of a state will actually be a quantitative parameter and will be subject to scaling. It is sometimes possible, however, to move from integer components to reals using the approach from [5].

The reduction from Π to P, according to Section 2, implies that the Pareto-optimal set of P can be efficiently transformed into a solution to the problem Π. Now let us suppose additionally that any $S \in \mathcal{S}_n$ can be mapped to some $y(S) \in Sol_{\mathbf{x}}$ and there is a function $G : \mathbb{N}_0^\beta \to \mathbb{N}_0$ such that $m_{\mathbf{x}}(y(S)) = G(S)$.

The assumption that the simplified DP algorithm described in Section 2 provides an exact solution to Π may be expressed formally:

$$OPT(\mathbf{x}) = \min\{G(S) : S \in \mathcal{S}_n\}, \tag{8}$$

22

if Π is a minimization problem, or alternatively

$$OPT(\mathbf{x}) = \max\{G(S) : S \in \mathcal{S}_n\}, \tag{9}$$

if Π is a maximization problem.

The function $y(S)$ is usually computed by means of a standard backtracking procedure (see e.g. [7], Chapter 15). A general description of such a procedure is beyond the scope of the paper since the details of reduction from problem Π to P are not considered here.

Suppose a *degree vector* $D = (d_1, \ldots, d_\beta) \in \mathbb{N}_0^\beta$ is defined for Π. Then, given a real value $\Delta > 1$ we say that $S = (s_1, \ldots, s_\beta)$ is (D, Δ)-*close* to $S' = (s'_1, \ldots, s'_\beta)$, if

$$\Delta^{-d_\ell} s_\ell \leq s'_\ell \leq \Delta^{d_\ell} s_\ell, \quad \ell = 1, \ldots, \beta.$$

Let us denote by $\mathcal{L}_0$ the set of indices $1 \leq \ell \leq \beta$ such that $d_\ell = 0$, and let $\mathcal{L}_1 = \{1, \ldots, \beta\} \backslash \mathcal{L}_0$.

The main tool to exclude unpromising states in a DP-based FPTAS [43] is the quasi-linear order $\preceq_{\text{qua}}$, which is an extension of a partial order $\preceq_{\text{dom}}$, i.e. if $S \preceq_{\text{dom}} S'$ then $S \preceq_{\text{qua}} S'$ for any $S, S' \in \mathbb{N}_0^\beta$. For the sake of compatibility with [43], we will limit the consideration to the case where $\preceq_{\text{dom}}$ is a partial order, rather than a more general partial quasi-order as in Sections 2 and 3. This restriction is not significant w.r.t. applications of the framework, although most likely the results of [43], as well as our results below, hold for the partial quasi-orders as well.

At each phase i, $i = 1, \ldots, n$, in DP_Δ only those states S may be excluded that are dominated in terms of $\preceq_{\text{qua}}$ by one of the other obtained states S', provided that S' is (D, Δ)-close to S.

Note that for any instance $\mathbf{x}$ the partial order $\preceq_{\text{dom}}$ on the final sets $\mathcal{S}_1, \ldots, \mathcal{S}_n$ may be represented by a finite number of criteria $g_1, \ldots, g_d$ of a corresponding instance of the problem P so that the Pareto-dominance relation is equivalent to $\preceq_{\text{dom}}$ on this set.

A problem Π is called *DP-benevolent* if besides C.1 and C.2, the following conditions C.1′,C.2′,C.3 and C.4 hold:

Condition C.1′. For any $\Delta > 1$, $S, S' \in \mathbb{N}_0^\beta$ and $F \in \mathcal{F}_i, i = 1, \ldots, n$, if S is (D, Δ)-close to S' and $S \preceq_{\text{qua}} S'$, then either $F(S) \preceq_{\text{qua}} F(S')$ and $F(S)$ is (D, Δ)-close to $F(S')$, or $F(S) \preceq_{\text{dom}} F(S')$.

Condition C.2′. For any $\Delta > 1$, $S, S' \in \mathbb{N}_0^\beta$ and $i = 1, \ldots, n$, if S is (D, Δ)-close to S' and $S \preceq_{\text{qua}} S'$, then $H_i(S') \leq H_i(S)$.

Condition C.3. A value $\gamma \in \mathbb{N}_0$ exists, depending only on G and D, such that for any $\Delta > 1$ and $S, S' \in \mathbb{N}_0^\beta$,

(i) if S is (D, Δ)-close to S' and $S \preceq_{\text{qua}} S'$, then $G(S') \leq \Delta^\gamma G(S)$ in the case of minimization, and $\Delta^{-\gamma} G(S) \leq G(S')$ in the case of maximization problem,

(ii) if $S \preceq_{\text{dom}} S'$, then $G(S') \leq G(S)$ in the case of minimization, and $G(S') \geq G(S)$ in the case of maximization problem.

Condition C.4.
(i) The functions $F \in \mathcal{F}_i$, $H_i, i = 1, \ldots, n$ and G, as well as the relation $\preceq_{\text{qua}}$ are computable in time polynomially bounded in the input length.
(ii) $|\mathcal{F}_i|$, $i = 1, \ldots, n$ is polynomially bounded in input length.
(iii) $\mathcal{S}_0$ is computable in time polynomially bounded in input length.
(iv) A polynomial $\pi_1(n, \log_2 |\mathbf{x}|)$ exists, such that all coordinates of any element $S \in \mathcal{S}_i$, $i = 1, \ldots, n$ are integer numbers bounded by $e^{\pi_1(n, \log_2 |\mathbf{x}|)}$. Besides that, for all $\ell \in \mathcal{L}_0$, the cardinality of the set of values that such a coordinate can take $|\{s_\ell : (s_1, \ldots, s_\ell, \ldots, s_\beta) \in \mathcal{S}_i\}|$ is bounded by a polynomial $\pi_2(n, \log_2 |\mathbf{x}|)$.

Example: knapsack problem. We can verify the DP-benevolence conditions for the knapsack problem as a simple illustrating example. Let the problem input, the DP states and the sets of mappings $\mathcal{F}_i$, $i = 1, \ldots, n$, as well as functions H_i be defined as in Section 2.3. Besides that, $G(S) \equiv s_2$ for all $S = (s_1, s_2) \in \mathcal{S}_n$ and the degree vector is $D = (1, 1)$.

A proper linear quasi-order $\preceq_{\text{qua}}$ that suits the partial order $\preceq_{\text{dom}}$ defined in Section 2.3 for the knapsack problem is not known to us. Instead, we can consider the following relations $\preceq_{\text{qua}}$ and $\preceq_{\text{dom}}$: let $S \preceq_{\text{qua}} S'$ iff $s_1 \geq s_1'$, where $S = (s_1, s_2)$, $S' = (s_1', s_2')$ and let $\preceq_{\text{dom}}$ be the trivial partial order, i.e. $S \preceq_{\text{dom}} S'$ iff $S = S'$. (For an example of a DP-benevolent problem with non-trivial $\preceq_{\text{dom}}$ see the problem of minimizing total late work on a single machine [43].)

The statements in Conditions C.1, C.2, and C.3(ii) are fulfilled since

$\preceq_{\mathrm{dom}}$ is trivial. The function $G(s_1, s_2) \equiv s_2$ satisfies Condition C.3(i), which can be verified straightforwardly, assuming $\gamma = 1$. To see that Condition C.4 holds, consider a polynomial $\pi_1(n, \log_2 |\mathbf{x}|) = \ln(2^{|\mathbf{x}|})$, which ensures that $\max\{s_\ell \in \mathcal{S}_i | \ell = 1, 2, \ i = 1, \dots, n\} \leq \max\{\sum_{i=1}^{n} p_i, W\} \leq 2^{|\mathbf{x}|} = e^{\pi_1(n, \log_2 |\mathbf{x}|)}$.

Conditions C.1' and C.2' hold because the functions F_i, id and H_i at any phase i just sum the arguments with given non-negative constants. Indeed, consider e.g. the function $F_i(s_1, s_2) = (s_1 + w_i, s_2 + p_i)$. Here for any $\Delta > 1$, if $s_\ell / \Delta \leq s'_\ell \leq \Delta s_\ell$, $\ell = 1, 2$, then $(s_1 + w_i)/\Delta \leq s'_1 + w_i \leq \Delta(s_1 + w_i)$ and $(s_2 + p_i)/\Delta \leq s'_2 + p_i \leq \Delta(s_2 + p_i)$, therefore $F_i(s_1, s_2)$ is (D, Δ)-close to $F_i(s'_1, s'_2)$. Besides that, adding a constant to s_1 does not change the order $\preceq_{\mathrm{qua}}$. The functions id and H_i are treated analogously.

The other problems considered in Section 2.3 either do not admit FPTAS unless P=NP (the TSP), or they are solvable in time which is polynomially bounded in the input length and thus do not require FPTAS (the SSSP and the APSP problems).

4.1. Fully polynomial-time approximation scheme

To identify subsets of states which are (D, Δ)-close to each other, the algorithm DP_Δ employs a partition of the set of states into Δ-*boxes* (defined below). This partition allows to discard "close" states analogously to discarding of $(1+\varepsilon)$-dominated solutions which is used in multi-objective optimization for approximation of Pareto-set (see e.g. [22]). The main difference is that in our case the states are compared on the basis of their components, rather than the components of the vector of objectives. Note that usage of a quasi-linear order $\preceq_{\mathrm{qua}}$ in DP_Δ will make (D, Δ)-closeness only a necessary condition for discarding states from consideration.

Let L be a sufficiently large value, chosen for $\mathbf{x}$ and for any required precision $\varepsilon \in (0, 1)$ (a specific definition of L will be discussed later). To describe the algorithm DP_Δ let us consider a family of parallelepipeds that constitute a partition of the set $B(L, \Delta) = \mathbb{N}_0^\beta \cap [0, \Delta^L]^\beta$:

$$\{\mathcal{B}_{(k_1, \dots, k_\beta)} : k_\ell = 0, \dots, L, \ \ell = 1, \dots, \beta\},$$

where $\mathcal{B}_{(k_1, \dots, k_\beta)}$ contains all integer points $S = (s_1, \dots, s_\beta) \in \mathbb{N}_0^\beta$, such that:

$$s_\ell \in \begin{cases} 0, & \text{if } k_\ell = 0, \\ [\Delta^{k_\ell - 1}, \Delta^{k_\ell} - 1], & \text{if } 0 < k_\ell < L, \\ [\Delta^{k_\ell - 1}, \Delta^{k_\ell}], & \text{if } k_\ell = L, \end{cases} \tag{10}$$

Algorithm 3 DP_Δ for Π
___
1: $\mathcal{T}_0 \leftarrow \mathcal{S}_0$
2: **for** $i = 1$ to n **do**
3:     $\mathcal{T}_i \leftarrow \emptyset$
4:     **for** $S \in \mathcal{T}_{i-1}$ and $F \in \mathcal{F}_i$ **do**
5:         let $\mathcal{B}_{(k_1,\dots,k_\beta)}$ be the Δ-box containing $F(S)$
6:         **if** $H_i(F(S)) \leq 0$ and $\nexists S' \in \mathcal{T}_i \cap \mathcal{B}_{(k_1,\dots,k_\beta)} \colon F(S) \preceq_{\mathrm{qua}} S'$ **then**
7:             $\mathcal{T}_i \leftarrow (\mathcal{T}_i \setminus \{S' \in \mathcal{T}_i \cap \mathcal{B}_{(k_1,\dots,k_\beta)} \mid S' \prec_{\mathrm{qua}} F(S)\}) \cup \{F(S)\}$
8:         **end if**
9:     **end for**
10: **end for**
11: find $S^* \in \mathcal{T}_n$ such that

$$G(S^*) = \begin{cases} \min\{G(S) : S \in \mathcal{T}_n\} & \text{in case of minimization,} \\ \max\{G(S) : S \in \mathcal{T}_n\} & \text{in case of maximization} \end{cases}$$

12: **return** $y(S^*)$
___

for all $\ell \in \mathcal{L}_1$ and

$$s_\ell = k_\ell,$$

for all $\ell \in \mathcal{L}_0$. Thus defined parallelepipeds are called Δ-boxes below.

Algorithm 3 was suggested in [43] where it was proven to constitute an FPTAS with Δ and L chosen as follows

$$\Delta = 1 + \frac{\varepsilon}{2\gamma n}, \tag{11}$$

$$L = \left\lceil \frac{\pi_1(n, \log_2 |\mathbf{x}|)}{\ln \Delta} \right\rceil. \tag{12}$$

Equations (11) and (12) ensure L is polynomially bounded in size of the input and in $1/\varepsilon$.

4.2. Fully Polynomial-Time Randomized Approximation Scheme

A family of randomized algorithms over all factors $0 < \varepsilon < 1$ with polynomially bounded running times in problem input size $|\mathbf{x}|$ and in $1/\varepsilon$ that computes $(1 + \varepsilon)$-approximate solutions with probability at least $3/4$ is called a *fully polynomial-time randomized approximation scheme* (FPRAS) [24]. The

constant 3/4 in the definition of FPRAS for optimization problems may be replaced by any other constant from the interval (0,1).

The DP-based EA framework proposed in Section 3 may be modified to obtain an evolutionary FPRAS for DP-benevolent problems.

Now a new relation $\preceq_\Delta$ is defined to substitute $\preceq_{\mathrm{dom}}$ in Algorithm 2. Let us introduce the following relation: $(i, S) \preceq_\Delta (i', S')$, iff $H_i(S) > 0$ or the following three conditions hold:

a) $i = i'$

b) there exist such $k_1, \ldots, k_\beta$ that $S, S' \in \mathcal{B}_{(k_1, \ldots, k_\beta)}$

c) $S \preceq_{\mathrm{qua}} S'$.

The EA using this relation is denoted EA_Δ in what follows.

For an arbitrary $S \in \mathcal{S}_i$ let $\theta(i, S)$, $i = 1, \ldots, n$, be the first iteration number, when an individual (i, T) was added into population, such that:

(i) T is (D, Δ^i)-close to S and

(ii) $S \preceq_{\mathrm{qua}} T$.

In all iterations following $\theta(i, S)$ the population will contain an individual T that satisfies the conditions (i) and (ii) as well.

The following lemma indicates that for any non-dominated $S \in \mathcal{S}_i$, in a number of iterations that is on average polynomially bounded in $|\mathbf{x}|$ and $1/\varepsilon$, an individual (i, T) will be obtained such that T is (D, Δ^i)-close to S and $S \preceq_{\mathrm{qua}} T$. The proofs of the lemma and the theorem below are provided in [16] but since this publication might be difficult to access, we reproduce the proofs here.

Lemma 1. *Let Π be DP-benevolent with dimension β. Then for any stage $i = 0, \ldots, n$, any non-dominated state S in $\mathcal{S}_i$ and L chosen as defined in Equation 12 it holds that*

$$E[\theta(i, S)] \leq n(L\pi_2(n, \log_2 |\mathbf{x}|))^\beta \cdot \sum_{k=1}^{i} |\mathcal{F}_k|.$$

Proof. Let us use induction on i. For $i = 0$ the statement holds trivially. Consider any state S which is non-dominated in $\mathcal{S}_i$. Suppose $i > 0$ and the statement holds for $i - 1$.

Lemma 4.7 in [43] implies that there exists a state $S^{\#}$ non-dominated in $\mathcal{S}_{i-1}$ and a mapping $F^{\#} \in \mathcal{F}_i$, such that $F^{\#}(S^{\#}) = S$. Note that the induction hypothesis gives an upper bound on expectation of $\theta(i - 1, S^{\#})$, which is the expected number of iterations until an individual $(i - 1, T^{\#})$ is obtained, such that $T^{\#}$ is (D, Δ^{i-1})-close to $S^{\#}$ and $S^{\#} \preceq_{\text{qua}} T^{\#}$. Again, let the mutation that applies $F^{\#}$ to an individual $(i - 1, T^{\#})$ be called a successful mutation.

In view of C.2$'$ condition,

$$H_i(F^{\#}(T^{\#})) \leq H_i(S) \leq 0,$$

and by C.1 and C.1$'$, either (a) $F^{\#}(T^{\#})$ is (D, Δ^{i-1})-close to S and $S \preceq_{\text{qua}} F^{\#}(T^{\#})$, or (b) $S \preceq_{\text{dom}} F^{\#}(T^{\#})$.

In case (a), after a successful mutation, the population will contain the element $(i, F^{\#}(T^{\#}))$, or some other element (i, T') such that T' belongs to the Δ-box $\mathcal{B}_{(k_1, \ldots, k_\beta)}$, which also contains $F^{\#}(T^{\#})$ and besides this $F^{\#}(T^{\#}) \preceq_{\text{qua}} T'$. After this mutation the population will contain an individual (i, T), such that T is (D, Δ)-close to $F^{\#}(T^{\#})$ and $F^{\#}(T^{\#}) \preceq_{\text{qua}} T$. Now since $F^{\#}(T^{\#})$ is (D, Δ^{i-1})-close to S, by the definition of closeness, T is (D, Δ^s)-close to S. Besides that, $S \preceq_{\text{qua}} F^{\#}(T^{\#}) \preceq_{\text{qua}} T$, consequently, $S \preceq_{\text{qua}} T$. Thus, in case (a), successful mutation ensures presence of the required representative for S in population on stage i.

In case (b), a successful mutation will yield the individual (i, S), since S is a non-dominated state, and $S \preceq_{\text{dom}} F^{\#}(T^{\#})$. After such a mutation, the population will contain an individual (i, T), such that T is (D, Δ)-close to S and $S \preceq_{\text{qua}} T$. Obviously, T is also (D, Δ^i)-close to S then.

To complete the proof it remains to estimate the expected number of mutation attempts θ^* until a successful mutation occurs, conditioned that an individual $a^{\#} = (i - 1, T^{\#})$ belongs to the current population $\mathcal{P}$. Note that the probability of a successful mutation is

$$p^* = \left(n \cdot |\{(i - 1, S') \in \mathcal{P}\}| \cdot |\mathcal{F}_i| \right)^{-1},$$

at the same time,

$$|\mathcal{P}| = \sum_{i'=1}^{n} |\{(i', S') \in \mathcal{P}\}| \leq \sum_{i'=1}^{n} |\{(k_1, \ldots, k_\beta) : \mathcal{B}_{(k_1, \ldots, k_\beta)} \cap \mathcal{S}_{i'} \neq \emptyset\}|. \quad (13)$$

Consider a single term in the right-hand side of inequality (13) with any fixed i'. For each $\ell \in \mathcal{L}_1$ the index k_ℓ may take at most L different values. Besides that, in view of condition C4 (iv), for each $\ell' \in \mathcal{L}_0$ the coordinate $s_{\ell'}$ characterizing the states from the set $\mathcal{S}_{i'}$ may take at most $\pi_2(n, \log_2 |\mathbf{x}|)$ values.

Thus, the right-hand side of inequality (13) can not exceed

$$nL^{|\mathcal{L}_1|}\pi_2(n, \log_2 |\mathbf{x}|)^{|\mathcal{L}_0|} \leq n(L\pi_2(n, \log_2 |\mathbf{x}|))^\beta.$$

The statement of the lemma for phase i follows from the fact that $E[\theta(i, S)] = E[\theta(i-1, T^\#)] + 1/p^*$. $\qquad\square$

The bound on $E[\theta(i, S)]$ obtained in Lemma 1 is used to choose the stopping criterion for the algorithm EA_Δ. Let the algorithm terminate after

$$\tau = 4n(L\pi_2(n, \log_2 |\mathbf{x}|))^\beta \sum_{i=1}^n |\mathcal{F}_i| \tag{14}$$

iterations.

Theorem 3. *If the problem Π is DP-benevolent, then the family of algorithms EA_Δ where Δ and L are chosen according to (11) and (12), using the stopping criterion (14) gives an FPRAS.*

Proof. In view of (8) and C3 (ii), there exists a non-dominated state $S^* \in \mathcal{S}_n$, such that $OPT(\mathbf{x}) = G(S^*)$. By Lemma 1, on average within at most $n(L\pi_2(n, \log_2 |\mathbf{x}|))^\beta \cdot \sum_{i=1}^n |\mathcal{F}_i|$ iterations of EA_Δ, a population will be computed, containing an individual (n, T^*), such that T^* is (D, Δ^n)-close to S^* and $S^* \preceq_{qua} T^*$.

Let us first consider the case where Π is a minimization problem. By condition C3 (i):

$$G(T^*) \leq \Delta^{\gamma n} G(S^*) = \left(1 + \frac{\varepsilon}{2\gamma n}\right)^{\gamma n} OPT(\mathbf{x}) \leq (1 + \varepsilon)OPT(\mathbf{x}).$$

The latter inequality follows from the observations that $\gamma n \geq 1$, $(1 + \frac{\varepsilon}{2\gamma n})^{\gamma n}$ is a convex function in ε on the interval $\varepsilon \in [0, 2]$, and the indicated inequality holds for both endpoints of this interval. In the case of maximization problem Π analogously we obtain $G(T^*) \geq (1 + \varepsilon)^{-1}OPT(\mathbf{x})$.

29

Finally, by means of backtracking, a $(1 + \varepsilon)$-approximate solution $y(T^*)$ may be computed efficiently.

Execution of EA_Δ with stopping criterion (14), according to the Markov inequality, does *not* yield a $(1 + \varepsilon)$-approximate solution with probability at most $1/4$.

Finally, by condition C.4, the runtime of each iteration of the EA_Δ is polynomially bounded in the input length and in $1/\varepsilon$. Summing up the observed facts, we conclude that the proposed family of the algorithms constitutes an FPRAS. $\qquad\square$

5. Conclusions

We have examined how to choose a representation for an evolutionary algorithm such that it obtains the ability to carry out dynamic programming. Based on a general framework for dynamic programming we have given a framework for evolutionary algorithms that have a dynamic programming ability and analyzed the optimization time of such an algorithm depending on the corresponding dynamic programming approach. By considering well-known combinatorial optimization problems, we have shown that our framework captures most of the known DP-based evolutionary algorithms and allows to treat other problems.

Acknowledgements

Appendix A. Bellman Principle for Single-Objective Problems

In this appendix, we describe the Bellman optimality principle in terms of the DP method defined by recurrence (1). Consider a single-objective maximization problem Π. Let $2 \leq \beta \in \mathbb{N}$ be a constant and $\mathcal{S} \subseteq \mathbb{R}^\beta$, so that the first component s_1 of a state $S \in \mathcal{S}$ characterizes a quality of the state in some sense.

The Bellman principle applies to a DP algorithm for Π if the following statement holds. Suppose that starting from some state $S_0^* \in \mathcal{S}_0$, a sequence of "decisions" $F_1 \in \mathcal{F}_1, \ldots, F_n \in \mathcal{F}_n$ leads to an optimal solution for Π. Let us denote $S_i^* = F_i(F_{i-1}(\ldots F_1(S_0^*)\ldots)) \in \mathcal{S}_i$, $i = 1, \ldots, n$. Then for any particular state $S_i^* = (s_{1i}^*, \ldots, s_{\beta i}^*)$, the subsequence $F_1, \ldots, F_i$ is an optimal policy for reaching the set of states coinciding with S_i^* in components $s_2, \ldots, s_\beta$. By an optimal policy here we mean that for any sequence $F_1' \in \mathcal{F}_1, \ldots, F_i' \in \mathcal{F}_i$ starting with some $S_0' \in \mathcal{S}_0$, such that $S_k' = F_k'(F_{k-1}'(\ldots F_1'(S_0')\ldots)) \in \mathcal{S}_k$, $k = 1, \ldots, i$, and $S_i' = (s_{1i}', s_{2i}^*, \ldots, s_{\beta i}^*)$, holds $s_{1i}' \leq s_{1i}^*$.

If the Bellman principle applies to a DP algorithm, then for any $s_2, \ldots, s_\beta$ it is possible to keep only one state which dominates all states in the subset $\{S' \in \mathcal{S}_i \ : \ s_2' = s_2, \ldots, s_\beta' = s_\beta\}$ without a risk to loose optimality of the DP algorithm result.

Appendix B. Genetic Mechanisms Corresponding to the Mutation Proposed in the EA

The mutation operator proposed in the EA in Section 3.2 is a special case of the point mutation, where a gene A_i subject to change is selected as the first gene which has never been mutated so far (i.e. the first "undefined" gene). Such type of mutation may be imagined in a biological system as follows.

Suppose that for each phase i, $i = 1, \ldots, n$, there is a "controlling" gene B_i. The required localization of mutations in gene A_i, when A_i is the first "undefined" gene, is caused by insertion of some mobile DNA sequence C_i (e.g. a transposon, see [40]), that can enter the locus of gene A_i, and only this locus. We can additionally assume that a mobile element C_i is produced if and only if the gene B_i is active (i.e. B_i is subject to transcription in the parent individual). Besides that, we can assume that gene A_i in the "undefined" condition is silencing the transcription of gene B_{i+1}, but any mutated state of gene A_i activates the transcription of gene B_{i+1} and silences the gene B_i.

Then one can assume that in the i-th generation, $i = 1, \ldots, n$, only the gene B_i is active among $B_1, \ldots, B_n$, provided that initially only the gene B_1 was active. At the same time, in the i-th generation, $i = 1, \ldots, n$, the insertion mutations occur only in the gene A_i.

In nature, an example of a mutually exclusive genes activation is observed in malaria parasite Plasmodium falciparum. The transitions from one variant

of a gene to another one depend on the currently active gene variant [23].

References

[1] A. Auger and B. Doerr, editors. *Theory of Randomized Search Heuristics: Foundations and Recent Developments.* World Scientific, 2011.

[2] G. Ausiello and M. Protasi. Local search, reducibility and approximability of NP-optimization problems. *Information Processing Letters*, 54(2):73–79, 1995.

[3] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming.* Princeton University Press, 1962.

[4] H.-G. Beyer, H.-P. Schwefel, and I. Wegener. How to analyze evolutionary algorithms. *Theoretical Computer Science*, 287(1):101–130, 2002.

[5] S. S. Chauhan, A. V. Eremeev, A. A. Romanova, V. V. Servakh, and G. J. Woeginger. Approximation of the supply scheduling problem. *Operations Research Letters*, 33(3):249–254, 2005.

[6] R. Chowdhury, L. Hai-Son, and V. Ramachandran. Cache-oblivious dynamic programming for bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7:495–510, 2010.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[8] B. Doerr, A. Eremeev, C. Horoba, F. Neumann, and M. Theile. Evolutionary algorithms and dynamic programming. In *Proceedings of the 11th Genetic and Evolutionary Computation Conference (GECCO)*, pages 771–777. ACM Press, 2009.

[9] B. Doerr, E. Happ, and C. Klein. Crossover can provably be useful in evolutionary computation. In *Proceedings of the 10th Genetic and Evolutionary Computation Conference (GECCO)*, pages 539–546. ACM Press, 2008.

[10] B. Doerr, N. Hebbinghaus, and F. Neumann. Speeding up evolutionary algorithms through asymmetric mutation operators. *Evolutionary Computation*, 15(4):401–410, 2007.

[11] B. Doerr and D. Johannsen. Adjacency list matchings — an ideal genotype for cycle covers. In *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO)*, pages 1203–1210. ACM Press, 2007.

[12] B. Doerr, C. Klein, and T. Storch. Faster evolutionary algorithms by superior graph representations. In *Proceedings of the 1st IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, pages 245–250. IEEE Press, 2007.

[13] B. Doerr and M. Theile. Improved analysis methods for crossover-based algorithms. In *Proceedings of the 11th Genetic and Evolutionary Computation Conference (GECCO)*, pages 247–254, 2009.

[14] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, 2nd edition, 2007.

[15] A. V. Eremeev. On linking dynamic programming and multi-objective evolutionary algorithms, 2008. Omsk State University. Preprint (In Russian).

[16] A. V. Eremeev. A fully polynomial randomized approximation scheme based on an evolutionary algorithm. *Diskretnyi Analiz i Issledovanie Operatsii*, 17(4):3–17, 2010. (In Russian).

[17] T. Friedrich, N. Hebbinghaus, F. Neumann, J. He, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. In *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO)*, pages 797–804. ACM Press, 2007.

[18] M. Garey and D. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[19] O. Giel and I. Wegener. Evolutionary algorithms and the maximum matching problem. In *Proceedings of the 20th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 415–426. Springer, 2003.

[20] D. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley, 1989.

[21] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[22] C. Horoba. Analysis of a simple evolutionary algorithm for the multi-objective shortest path problem. In *Proceedings of 10th International Workshop on Foundations of Genetic Algorithms (FOGA), Orlando, Florida, USA*, pages 113–120, New York, NY, USA, 2009. ACM Press.

[23] P. Horrocks, R. Pinches, Z. Christodoulou, S. Kyes, and C. Newbold. Variable var transition rates underlie antigenic variation in malaria. *Proceedings of the National Academy of Sciences, USA*, 101:11129–11134, 2004.

[24] M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22(5):1087–1116, 1993.

[25] R. Klötzler. Multiobjective dynamic programming. *Mathematische Operationsforschung und Statistik. Series Optimization*, 9:423–426, 1978.

[26] M. Laumanns, L. Thiele, E. Zitzler, E. Welzl, and K. Deb. Running time analysis of multi-objective algorithms on a simple discrete optimization problem. In *Proceedings of the 7th International Conference on Parallel Problem Solving From Nature (PPSN VII)*, volume LNCS 2439, pages 44–53. Springer, 2002.

[27] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics.* Springer, 2004.

[28] L. G. Mitten. Composition principles for synthesis of optimal multistage processes. *Operations Research*, 12(4):610–619, 1964.

[29] F. Neumann. Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. *Computers and Operations Research*, 35(9):2750–2759, 2008.

[30] F. Neumann and J. Reichel. Approximating minimum multicuts by evolutionary multi-objective algorithms. In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN)*, pages 72–81. Springer, 2008.

[31] F. Neumann, J. Reichel, and M. Skutella. Computing minimum cuts by randomized search heuristics. In *Proceedings of the 10th Genetic and Evolutionary Computation Conference (GECCO)*, pages 779–786. ACM Press, 2008.

[32] F. Neumann and M. Theile. How crossover speeds up evolutionary algorithms for the multi-criteria all-pairs-shortest-path problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN XI)*, volume LNCS 6238, pages 667–676. Springer, 2010.

[33] F. Neumann and I. Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40, 2007.

[34] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.

[35] C. Potts and M. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

[36] G. R. Raidl and B. A. Julstrom. Edge sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3):225–239, 2003.

[37] J. Reichel and M. Skutella. Evolutionary algorithms and matroid optimization problems. In *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO)*, pages 947–954. ACM Press, 2007.

[38] F. Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer-Verlag, 2006.

[39] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366, 2004.

[40] D. J. Sherratt. *Mobile Genetic Elements*. Oxford University Press, 1995.

[41] M. Theile. Exact solutions to the traveling salesperson problem by a population-based evolutionary algorithm. In *Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)*, volume 5482 of *LNCS*, pages 145–155. Springer, 2009.

[42] C. Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 44–56. Springer, 2005.

[43] G. J. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12(1):57–74, 2000.