Evolutionary Algorithms and Dynamic Programming

Benjamin Doerr
Algorithms and Complexity
Max-Planck-Institut für Informatik
Saarbrücken, Germany

Anton Eremeev
Omsk Branch of Sobolev Institute of Mathematics
Omsk, Russia

Christian Horoba
Fakultät für Informatik, LS 2
TU Dortmund
Dortmund, Germany

Frank Neumann
Algorithms and Complexity
Max-Planck-Institut für Informatik
Saarbrücken, Germany

Madeleine Theile
Institut für Mathematik
TU Berlin
Berlin, 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.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Design, Performance, Theory

Keywords

Combinatorial optimization, dynamic programming, evolutionary algorithms

1. INTRODUCTION

Evolutionary algorithms have been shown to be successful for a wide range of optimization problems. While these randomized search heuristics work well for many optimization problems in practice, a satisfying and rigorous mathematical understanding of their performance is an important challenge in the area of genetic and evolutionary computing.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

Interesting results have been obtained for some important optimization problems like sorting and shortest paths [18], spanning trees [15], maximum matchings [7], and minimum cuts [13, 14]. There are also some results on evolutionary algorithms acting as approximation algorithms for NP-hard problems like partition [20], covering [6], and multi-objective shortest path [9] problems. But a general theoretical explanation of the behavior of evolutionary algorithms is still missing. The first step in this direction is taken in [17], where the authors show for an important subclass of optimization problems, that evolutionary algorithms permit optimal solutions in polynomial time.

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. 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, for example there are different representations for the well-known traveling salesman problem (see e. g. Michalewicz [11]) or NP-hard spanning tree problems (see e. g. Raidl and Julstrom [16]).

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 all-pairs shortest path problem [2] and the traveling salesman problem [19]. 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 that allow the algorithms to proceed in a dynamic programming way.

Dynamic programming [1] 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 [21] and Kogan [10]. The technique allows the design of efficient algorithms, that solve the problem at hand to optimality, by extending partial solutions to optimal ones. The goal of this paper is to relate the evolutionary approaches given in [2, 19] 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 any problem that can be solved by dynamic programming in time T has an evolutionary algorithm which solves the problem in expected time $O(T \cdot n \cdot \log(|DP|))$ with n being the number of stages and $|DP|$ being the number of states produced in the dynamic programming.

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. In Section 4, we show how to obtain evolutionary algorithms carrying out dynamic programming for some well-known combinatorial optimization problems. Finally, we finish with some conclusions.

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 original 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. For some problems it is even the most efficient approach known for the solution.

2.1 Considered Problems

Consider a multi-objective optimization problem P with $k \in \mathbb{N}^+$ being the number of objectives that have to be optimized. We present such an optimization problem as a function $g\colon \mathcal{S} \to \mathbb{R}^k$ where $\mathcal{S}$ is called *search space*, g is called *objective function*, and $g(\mathcal{S}) \subseteq \mathbb{R}^k$ is the *objective space*. Note that single-objective optimization is a special case of multi-objective optimization where the number of objectives is 1.

For a single-objective optimization problem the goal is to determine a solution with an optimal objective value where the total order $\leq$ or $\geq$ on the objective space is considered. For a multi-objective optimization problem there is no natural total order on the objective space. Hence, a natural partial order is considered instead, which implies that there is no single optimal objective vector but a set of optimal objective vectors.

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

$$(y'_1, \ldots, y'_k) \succeq (y_1, \ldots, y_k)$$

iff $y'_i \leq y_i$ for all i for minimization problems or $y'_i \geq y_i$ for all i for maximization problems. In the following, we use the notation $y' \succ y$ as an abbreviation for $y' \succeq y$ and $y \not\succeq y'$. The *Pareto front* is the subset of $g(\mathcal{S})$ that consists of all maximal elements of $g(\mathcal{S})$ with respect to $\succeq$. The goal is to determine a *Pareto-optimal set*, that is, a minimal subset of $\mathcal{S}$ that is mapped on the Pareto front.

2.2 Framework for Dynamic Programs

Consider a Dynamic Program (DP) for an optimization problem P. Suppose, the DP produces new partial solutions in n phases, such that in the i-th phase, a set $\mathcal{S}_i \subseteq \mathcal{S}$ of partial solutions is created. We use n finite sets $\mathcal{F}_i$ of mappings $F\colon \mathcal{S} \to \mathcal{S}'$ and a mapping $H\colon \mathcal{S}' \to \mathbb{R}$ with $S \in \mathcal{S}$ if $H(S) \leq 0$ to describe the DP where $\mathcal{S}' \supseteq \mathcal{S}$ is an extension of the original search space $\mathcal{S}$. This is a necessary modification because a mapping F can create a new state that does not correspond to a feasible solution. We assume that n and all $\mathcal{F}_i$ depend on the length of the input instance and H depends on the input instance. We call the various F *state transition functions* and the H *consistency function*.

The DP proceeds as follows. In the initialization phase, the state space $\mathcal{S}_0$ is initialized with a finite subset of $\mathcal{S}$. In phase i, the elements of the state space $\mathcal{S}_{i-1}$ are considered as partial solutions to P, which can be extended to a feasible solution of P. In the i-th phase, the state space $\mathcal{S}_i$ is computed using the state space $\mathcal{S}_{i-1}$ according to

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

In the process, the consistency function H serves to keep the infeasible partial solutions emerging in phase i from being included into the current state space $\mathcal{S}_i$.

To improve the runtime, most DPs utilize the Bellman principle (see e. g. [1]). The basic idea is to dismiss unpromising partial solutions without affecting the optimality of the final set of solutions. We describe this principle as a dominance relation among the states in each state space $\mathcal{S}_i$.

We call $\mathrm{EXT}_k(S) \in \mathcal{S}_n$ a *feasible extension* of state $S \in \mathcal{S}_k$ if there are $F_i \in \mathcal{F}_i$, $i = k-1, \ldots, n$ with

$$\mathrm{EXT}_k(S) = F_n(F_{n-1}(\ldots F_{k+1}(S)\ldots))$$

and

$$H(F_i(F_{i-1}(\ldots F_{k+1}(S)\ldots))) \leq 0$$

for all $i = k+1, \ldots, n$. Consider two states $S, S' \in \mathcal{S}_k$. Assume that for all feasible extensions $\mathrm{EXT}_k(S)$ of state S to a complete solution there is a feasible extension $\mathrm{EXT}_k(S')$ of state S' to a complete solution where $g(\mathrm{EXT}_k(S'))$ is at least as good as $g(\mathrm{EXT}_k(S))$, that is, $g(\mathrm{EXT}_k(S')) \succeq g(\mathrm{EXT}_k(S))$. In this case, we say that state $S \in \mathcal{S}_k$ is *dominated in extension* by state $S' \in \mathcal{S}_k$. A state without any feasible extensions is dominated in extension by all states by definition. Such a situation allows to dismiss S without affecting the optimality of the final set of solutions.

Straightforward verification of this relation by checking all possible extensions is very inefficient. To ease the exclusion of unpromising states, we consider another partial order $\preceq_{\mathrm{dom}}$ defined on $\mathcal{S}$. We will see, that a partial order $\succeq_{\mathrm{dom}}$ that induces dominance in extension can be used to dismiss unpromising solutions. We say that state S is *dominated* by

state S' iff $S \preceq_{\text{dom}} S'$. 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 $\succeq_{\text{dom}}$ iff for all states $S \in \mathcal{S}_i$ there is a state $S' \in \mathcal{T}_i$ with $S \preceq_{\text{dom}} S'$. We use the notation $M(\mathcal{S}_i, \succeq_{\text{dom}})$ to denote the set of all dominating subsets of $\mathcal{S}_i$ with minimum cardinality.

It is crucial that the partial order $\preceq_{\text{dom}}$ can be checked efficiently. Hence, we are content with partial orders $\preceq_{\text{dom}}$ that are induced via $S \preceq_{\text{dom}} S'$ iff $f(S) \preceq_{\text{par}} f(S')$ using a function $f\colon \mathcal{S} \to \mathbb{R}^d$ and a partial order $\preceq_{\text{par}}$ on $\mathbb{R}^d$. We assume that f can be evaluated efficiently and $\preceq_{\text{par}}$ can be decided efficiently.

The following proposition indicates that it is sufficient to keep a dominating subset of states constructed in each phase k, rather than the full subsets $\mathcal{S}_k$.

PROPOSITION 1. *Let the dominating sets $\mathcal{T}_k$ be computed according to*

$$\mathcal{T}_k \in M(\{F(S) \mid S \in \mathcal{T}_{k-1} \wedge F \in \mathcal{F}_k \wedge H(F(S)) \leq 0\}, \succeq_{\text{dom}}).$$

If dominance implies dominance in extension then

$$g(\mathcal{T}_n) \in M(g(\mathcal{S}_n), \succeq).$$

Proof. Suppose the contrary. The equation (1) finally yields a state corresponding to an optimum, so there exist such $k \in \{1, \dots, n\}$ and $S \in \mathcal{S}_k \setminus \mathcal{T}_k$, that some extension $\text{EXT}_k(S)$ is optimal, and, at the same time, no extension $\text{EXT}_k(S')$ with $S' \in \mathcal{T}_k$ gives an optimum. But by definition, the dominating set $\mathcal{T}_k$ contains some element S', dominating S. Consequently, S' also dominates S in extensions and $g(\text{EXT}_k(S'))$ is optimal as well. This gives a contradiction. □

Note that the size of the $\mathcal{T}_i$ is uniquely determined. The last proposition implies that if the Pareto front of g is contained in $g(\mathcal{S}_n)$, it is also contained in $g(\mathcal{T}_n)$.

We consider in addition the following three conditions that express desirable properties of dominance relations.

CONDITION 1 (C.1). *For any $S, S' \in \mathcal{S}_i$, if $S \preceq_{\text{dom}} S'$ then $F(S) \preceq_{\text{dom}} F(S')$ for all $F \in \mathcal{F}_{i+1}$.*

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

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

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

CONDITION 3 (C.3). *For any $S, S' \in \mathcal{S}_n$, if $S \preceq_{\text{dom}} S'$ then $g(S) \preceq g(S')$.*

The last condition C.3 establishes a connection between the dominance relation and the objective function g that has to be optimized. It states that after the last round dominance $S \preceq_{\text{dom}} S'$ of states S, S' implies Pareto dominance $g(S) \preceq g(S')$ of the corresponding function values $g(S)$, $g(S')$.

The next proposition shows, that $\preceq_{\text{dom}}$ implies dominance in extension if C.1–C.3 are fulfilled.

PROPOSITION 2. *Suppose that C.1, C.2, and C.3 hold for $\preceq_{\text{dom}}$. Then for any $k \in \{0, \dots, n\}$ and $S, S' \in \mathcal{S}_k$, such that $S \preceq_{\text{dom}} S'$, the state S' also dominates S in extensions in phase k.*

Algorithm 1 Dynamic Program for g

```
1: T_0 ← ∅
2: for S ∈ S_0 do
3:    if ∄S' ∈ T_0: S' ≽_dom S then
4:       T_0 ← (T_0 \ {S' ∈ T_0 | S ≻_dom S'}) ∪ {S}
5:    end if
6: end for
7: for i = 1 to n do
8:    T_i ← ∅
9:    for S ∈ T_{i-1} and F ∈ F_i do
10:      if H(F(S)) ≤ 0 and ∄S' ∈ T_i: S' ≽_dom F(S) then
11:         T_i ← (T_i \ {S' ∈ T_i | F(S) ≻_dom S'}) ∪ {F(S)}
12:      end if
13:   end for
14: end for
15: return {S ∈ T_n | ∄S' ∈ T_n: g(S') ≻ g(S)}
```

Proof. If in phase k the state S has no feasible extensions then the required result follows trivially.

Suppose, S has a feasible extension

$$\text{EXT}_k(S) = F_n(F_{n-1}(\dots F_{k+1}(S)\dots)).$$

Then $n-k$ applications of condition C.1 produce

$$F_n(F_{n-1}(\dots F_{k+1}(S)\dots)) \preceq_{\text{dom}} F_n(F_{n-1}(\dots F_{k+1}(S')\dots)).$$

Furthermore, $n-k$ applications of condition C.2 lead to $H(F_i(\dots F_{k+1}(S')\dots)) \leq 0$ for all $i = k+1, \dots, n$ since $H(F_i(\dots F_{k+1}(S)\dots)) \leq 0$. Hence, S' has a feasible extension

$$\text{EXT}_k(S') = F_n(F_{n-1}(\dots F_{k+1}(S')\dots))$$

with $\text{EXT}_k(S) \preceq_{\text{dom}} \text{EXT}_k(S')$. Finally, $g(\text{EXT}_k(S'))$ is at least as good as $g(\text{EXT}_k(S))$ due to condition C.3. □

Expressing the computation according to the equation within Proposition 1 in algorithmic form, the general scheme of a DP for P is presented in Algorithm 1.

We call a DP *correct* if the Pareto front of g is contained in $g(\mathcal{S}_n)$ and $\preceq_{\text{dom}}$ implies dominance in extension.

The runtime of a DP depends on the computation time $\theta_{\mathcal{F}}$ for the state transition functions $F \in \mathcal{F}_i$, the computation time $\theta_{\mathcal{H}}$ for the consistency function H, and the computation time θ_{dom} for the dominance relation $\succeq_{\text{dom}}$. Let $\kappa := \max_{1 \leq i \leq n} |\mathcal{F}_i|$. The body (lines 10–12) of the main loop (lines 7–14) in Algorithm 1 is executed $\sum_{i=1}^{n} |\mathcal{T}_{i-1}| \cdot |\mathcal{F}_i| \leq \kappa \cdot \sum_{i=1}^{n} |\mathcal{T}_{i-1}|$ times. Hence, considering a straightforward implementation of Algorithm 1, the computation time for the main loop is $O\big(\kappa \cdot \sum_{i=1}^{n} |\mathcal{T}_{i-1}| \cdot (\theta_{\mathcal{F}} + \theta_{\mathcal{H}} + |\mathcal{T}_i| \cdot \theta_{\text{dom}})\big)$. 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 of

$$O\left(\theta_{\text{ini}} + \kappa \cdot \sum_{i=1}^{n} |\mathcal{T}_{i-1}| \cdot (\theta_{\mathcal{F}} + \theta_{\mathcal{H}} + |\mathcal{T}_i|\theta_{\text{dom}}) + \theta_{\text{out}}\right). \quad (2)$$

Note that the runtime of the DP is polynomially bounded in the input length if θ_{ini}, n, $|\mathcal{T}_i|$ for $i = 0, \dots, n$, κ, $\theta_{\mathcal{F}}$, $\theta_{\mathcal{H}}$, θ_{dom}, and θ_{out} are polynomially bounded in the input length. Furthermore, for $d \leq 2$ the dominance check can be sped up to take at most $\log(|T_i|) \cdot \theta_{\text{dom}}$ comparisons by sorting the entries of the objective vector according to the first coordinate.

Algorithm 2 Evolutionary Algorithm for g

$\mathcal{P} \leftarrow \emptyset$
for $I \in \mathcal{P}_0$ **do**
  if $\nexists I' \in \mathcal{P} \colon I \prec_{\text{dom}} I'$ **then**
    $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{I' \in \mathcal{P} \mid I' \preceq_{\text{dom}} I\}) \cup \{I\}$
  end if
end for
loop
  $I \leftarrow \text{mut}(\text{sel}(\mathcal{P}))$
  if $\nexists I' \in \mathcal{P} \colon I \prec_{\text{dom}} I'$ **then**
    $\mathcal{P} \leftarrow (\mathcal{P} \setminus \{I' \in \mathcal{P} \mid I' \preceq_{\text{dom}} I\}) \cup \{I\}$
  end if
end loop
return $\{\text{out}(I) \mid I \in \mathcal{P} \wedge \nexists I' \in \mathcal{P} \colon g(\text{out}(I)) \prec g(\text{out}(I'))\}$

3. EVOLUTIONARY ALGORITHMS

In the following, we show how dynamic programming can be carried out by evolutionary algorithms. To this aim, we state a general formulation of 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. [12, 3, 5, 4]), demonstrate that the right choice of representation, variation operators, and selection method is crucial for the success of such algorithms.

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

The algorithm (see Algorithm 2) starts with an initial population $\mathcal{P}_0$. During the optimization the evolutionary algorithm uses a selection operator $\text{sel}(\cdot)$ and a mutation operator $\text{mut}(\cdot)$ to create new individuals. The d-dimensional fitness function together with a partial order $\succeq_{\text{par}}$ on $\mathbb{R}^d$ induce a partial order $\succeq_{\text{dom}}$ on the phenotype space, which guides the search. After the termination of the EA, an output function $\text{out}(\cdot)$ is utilized to map the individuals in the last population to search points from the original search space.

3.2 Defining the Modules

We now consider how the different modules of the evolutionary algorithm have to be implemented such that it can carry out dynamic programming. To do this, we relate the modules to the different components of a DP. Consider a problem given by a multi-objective function g that can be solved by a dynamic programming approach. We use the indices DP and EA to differentiate between the components of the DP and the EA. The EA works with the following setting.

We use $\mathcal{S}'_{\text{EA}} := \{0, \dots, n\} \times \mathcal{S}'_{\text{DP}}$ as the phenotype space. The initial population is $\mathcal{P}_0 = \{0\} \times \mathcal{S}_0$ where $\mathcal{S}_0$ is the initial state space of the DP. The selection operator $\text{sel}(\cdot)$ chooses an individual $I \in \mathcal{P}$ uniformly at random for mutation. 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 use the fitness function $f_{\text{EA}} \colon \mathcal{S}'_{\text{EA}} \to \{0, \dots, n\} \times (\mathbb{R} \cup \{\pm\infty\})^d$ with

$$f_{\text{EA}}((i, S)) = \begin{cases} (i) \circ f_{\text{DP}}(S) & \text{if } H(S) \leq 0 \\ (i) \circ (\pm\infty)^d & \text{if } H(S) > 0 \end{cases}$$

where $\circ$ denotes the concatenation of two vectors. The meaning of the fitness function is as follows. If an individual (i, S) is feasible, that is, $H(S) \leq 0$, it is mapped on $(i, y_1, \dots, y_d)$ where $(y_1, \dots, y_d) = f_{\text{DP}}(S)$. Otherwise, it is mapped on $(i, \pm\infty, \dots, \pm\infty)$ where $y \succeq^{\text{DP}}_{\text{par}} (\pm\infty, \dots, \pm\infty)$ for all $y \in f_{\text{DP}}(\mathcal{S})$, which means that the individual is worse than all feasible individuals. We incorporate the partial order $\succeq^{\text{EA}}_{\text{par}}$ that is defined as $(i', y') \succeq^{\text{EA}}_{\text{par}} (i, y)$ iff $i' = i$ and $y' \succeq^{\text{DP}}_{\text{par}} y$ into the EA to guide the search. Finally, we utilize the output function $\text{out}_{\text{EA}}((i, S)) = S$. At the end of a run of the EA, the output function is needed to remove the additional information that is 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 that has to be solved.

3.3 Runtime of the Evolutionary Algorithm

Our goal is to show that the evolutionary algorithm solves the problem given by g efficiently if the dynamic programming approach does. To measure the time the evolutionary algorithm needs to compute an optimal solution for the given problem, we analyze the expected number of fitness evaluations to come up with an optimal solution. This is also called the expected *optimization time* of the considered algorithm, which is a common measure for analyzing the runtime behavior of evolutionary algorithms.

The next theorem relates the expected optimization time of the EA to the runtime of the corresponding DP.

THEOREM 1. *Let a DP be defined as in Algorithm 1 and an EA defined as in Algorithm 2. Then the DP has a runtime of*

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

and the EA has an expected optimization time of

$$O\left(\kappa \cdot \sum_{i=1}^{n} |\mathcal{T}_{i-1}| \cdot n \cdot \log\left(\sum_{i=1}^{n} |\mathcal{T}_{i-1}|\right)\right).$$

Proof. For the proof of the optimization time we will make some pessimistic assumptions on how the process works. Although the population starts with $\mathcal{P}_0$ we will assume that it has size $|\mathcal{P}| = \sum_{i=0}^{n} |\mathcal{T}_i| =: |DP|$ right from the start, where by $|DP|$ we denote the cardinality of the set of states produced by the DP. Furthermore assume that the optimization process works in stages $1 \leq i \leq n$, whereas stage $i + 1$ starts after stage i has been finished. Stage i finishes after every non-dominated $S \in \mathcal{T}_i$ has become part of the population $\mathcal{P}$. Thus, in stage $i + 1$ every Pareto-optimal individual $S \in \mathcal{T}_i$ can be used for the selection and mutation to an individual in $\mathcal{T}_{i+1}$. Consider the event that a non-dominated individual $(i + 1, S)$ with $S \in \mathcal{T}_{i+1}$ is created by the EA. This implies that no other $(i + 1, S')$ with $S' \in \mathcal{T}_{i+1}$ and $f((i + 1, S')) \succeq_{\text{par}} f((i + 1, S))$ is part of population $\mathcal{P}$. Obviously there exists an individual (i, S'') with $S'' \in \mathcal{T}_i$ and a $F \in \mathcal{F}_{i+1}$ such that $(i + 1, S') = \text{mut}((i, S''))$.

Let X_i be the random variable denoting the number of iterations until stage i is completed. Then the optimization time is given by $E[X]$ with $X = X_1 + \dots X_n$.

The probability for a successful mutation in stage $i+1$ depends on the current size of the population $|\mathcal{P}| \leq |DP|$ and the number of state transition functions $\mathcal{F}_i$. Let t denote the number of non-dominated individuals that have been added at stage $i+1$. Then there are $|\mathcal{T}_{i+1}| - t$ possibilities to add a new non-dominated individual. The probability for a successful mutation is thus no less than $(|\mathcal{T}_{i+1}| - t)/(|DP| \cdot |\mathcal{F}_{i+1}|)$ with an expected waiting time of at most $(|DP| \cdot |\mathcal{F}_{i+1}|)/(|\mathcal{T}_{i+1}| - t)$ for this geometrically distributed variable. The expected waiting time to finish stage $i+1$ is thus given by

$$E[X_{i+1}] \leq \sum_{t=1}^{|\mathcal{T}_{i+1}|} \frac{|DP| \cdot |\mathcal{F}_{i+1}|}{t} = |DP| \cdot |\mathcal{F}_{i+1}| \cdot H_{|\mathcal{T}_{i+1}|},$$

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

This leads to an overall expected number of iterations

$$\begin{aligned} E[X] &\leq \sum_{i=0}^{n-1} |DP| \cdot |\mathcal{F}_{i+1}| \cdot H_{|\mathcal{T}_{i+1}|} \\ &\leq \kappa \cdot |DP| \cdot \sum_{i=0}^{n-1} H_{|\mathcal{T}_{i+1}|} \\ &\leq \kappa \cdot |DP| \cdot n \cdot (\ln(|DP|) + 1). \end{aligned}$$ □

A similar inspection as in Subsection 2.2 reveals that for $|DP| = \sum_{i=1}^{n} |T_{i-1}|$ the expected runtime of the EA is

$$O\Big(\theta_{\text{ini}} + \kappa \cdot n \cdot \log(|DP|) \cdot \sum\nolimits_{i=1}^{n} |T_{i-1}| \cdot (\theta_{\mathcal{F}} + \theta_{\mathcal{H}} + |T_i| \cdot \theta_{\text{dom}}) + \theta_{\text{out}}\Big)$$

if the individuals of the population are stored in $n+1$ disjoint sets according to the first coordinate i, and

$$O\Big(\theta_{\text{ini}} + \kappa \cdot n \cdot \log(|DP|) \cdot |DP| \cdot (\theta_{\mathcal{F}} + \theta_{\mathcal{H}} + |DP| \cdot \theta_{\text{dom}}) + \theta_{\text{out}}\Big)$$

if the dominance check is done against the whole population.

4. EXAMPLES

In this section, we point out how the framework presented in the previous section can be used to construct evolutionary algorithms that have the ability of simulating dynamic programming for well-known combinatorial optimization problems. The approach followed here is to describe the components of a dynamic programming algorithm for the solution of a given problem. The existence and functionality of an evolutionary algorithm for the solution of the problem follows from Section 3.

4.1 All-Pairs Shortest Path Problem

A classical problem that fits into the DP framework is the all-pairs shortest path (APSP) problem. Given an undirected connected graph $G = (V, E)$ and positive edge weights $w\colon E \to \mathbb{R}^+$, the task is to find for each pair (u, v) of vertices a shortest path connecting them.

In [2] it has been shown that the problem is solved by evolutionary algorithms that follow the idea of dynamic programming in expected polynomial time. In the following, we discuss the basic ideas and show how the algorithm fits into our framework.

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, since the partial solutions that often are added artificially to the search space, here are an integral part of it.

For the APSP, the search space $\mathcal{S}$ naturally is the set of all paths in G. We model paths via finite sequences of vertices, and do not allow cycles. Since adding an edge to a path may create a cycle, which is an infeasible solution, let us extend this search space to the set $\mathcal{S}'$ of all sequences of vertices of length at most n. The set 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 $\mathcal{F}_i := \{F_v \mid v \in V\} \cup \{\text{id}\}$, where $F_v\colon \mathcal{S}' \to \mathcal{S}'$ is the mapping adding the vertex v to a sequence of vertices. To exclude invalid solutions, let us define $H(s)$ to be -1, if s is a path in G, and 1, if not. We can omit using the out-operator here.

It remains to define when one solution dominates another. This can be done in the framework of multi-objective optimization by defining a relatively artificial multi-objective objective function. A more natural way is to put $s \preceq s'$ if and only if the paths s and s' connect the same two vertices and s' is not longer than s.

Since the length of the path arising from extending an existing path by an edge depends monotonically on the length of the existing graph, the Bellman principle is easily correctly implemented through the natural dominance relation which lets s' dominate s if and only if $s \preceq s'$. In consequence, our solution set contains at most one path for each 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. 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 same end vertices, it replaces the latter. Apart from implementation issues (namely rather only storing the length of the paths instead of storing the whole paths), this is a variant of the well-known Floyd-Warshall algorithm. Plugging these ideas into the framework of Algorithm 2, we obtain an EA with an expected runtime of $O(n^4 \log(n))$ due to Theorem 1. This bound can be further improved to $\Theta(n^4)$ as has been shown in [2].

4.2 Traveling Salesman Problem

In the following, we consider the traveling salesman problem (TSP) problem as a prominent NP-hard example.

The input for the TSP consists of a complete graph $G = (V, E)$ with a set of nodes $V = \{1, 2, \dots, n\}$ and non-negative edge weights $w\colon E \to \mathbb{R}_0^+$. It is required to find a permutation of all nodes $(v_1, \dots, v_n)$, such that the TSP tour length $\sum_{i=2}^{n} w(v_i, v_{i-1}) + w(v_n, v_1)$ is minimized. Without loss of generality we can assume that $v_1 = 1$, that is, the TSP tour starts in the fixed vertex 1.

In [19] an evolutionary algorithm with dynamic program-

ming features has been introduced. In the following, we show how the algorithm fits into our framework. The search space $\mathcal{S}$ used in the dynamic programming algorithm of Held and Karp [8] consists of all pairs (i, M) with $M \subseteq V$ and $i \in M$. $\mathcal{S}'$ is the extended search space of all sequences of nodes up to length n. A pair (i, M) represents a Hamiltonian path starting in vertex 1 then running over all nodes from M and ending in vertex i. The initial population $\mathcal{P}_0$ consists of $n-1$ elements: for each $i = 2, \ldots, n$ there is an individual $(i, \{i\})$.

The set $\mathcal{F}_i$ for all i consists of $n-1$ functions: $\mathcal{F}_i = \{\mathrm{id}\} \cup \{F_v \mid v \in V\}$ with $F_v \colon \mathcal{S}' \to \mathcal{S}'$ being the mapping that adds vertex v to the existing sequence of vertices id. For invalid solutions, which are characterized by not being Hamiltonian paths on $F_v(s)$ with $s \in \mathcal{S}'$, the mapping $H(s)$ computes 1 and 0 otherwise.

Let us define the dominance relation $s \preceq s'$ if and only if s and s' are Hamiltonian paths on the same ground set M with the same end vertex i and path s' is not longer than s. Consequently, there is always only one individual for a specific (i, M) in the population. Substituting these components with the components in Algorithm 1, we see that we get the well-known dynamic programming algorithm of Held and Karp [8]. Algorithm 1 initializes the states of the dynamic program with the shortest possible optimal Hamiltonian paths $(v, \{v\})$ 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 $H(F(s))$ is a feasible partial solution, that is not dominated. If it is not dominated 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.

Due to Theorem 1 the expected optimization time of the evolutionary algorithm is $O\big(n^3 \cdot \log(n) \cdot 2^n\big)$. This bound can be further improved to $O\big(n^3 \cdot 2^n\big)$ as shown in [19].

4.3 Knapsack Problem

We have already seen that the known evolutionary algorithms for the APSP [2] and the TSP [19] fit into the proposed framework. Another well-known combinatorial optimization problem that can be solved by dynamic programming is the knapsack problem. We consider this problem as a third example on how to obtain evolutionary algorithms that have the additional ability of carrying out dynamic programming steps. The input for the knapsack problem consists of n items where each item i has an associated weight w_i and profit p_i, $1 \le i \le n$. 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 $\sum_{i \in K} w_i \le W$.

We fit the problem into above framework using the search space

$$\mathcal{S} := \left\{ x \in \{0,1\}^n \mid \sum_{i=1}^{n} x_i \cdot w_i \le W \right\}$$

and the pseudo-Boolean function $g \colon \mathcal{S} \to \mathbb{R}$ with

$$g(x) := \sum_{i=1}^{n} x_i \cdot p_i$$

that has to be maximized.

The classical DP for the knapsack problem works as follows. It starts with the state space $\mathcal{S}_0 = \{0^n\}$ containing a state that encodes the selection of no items. The DP runs through n rounds. In each round it tries to add the i-th item to all item selections known so far. Formally, we have $\mathcal{F}_i = \{F_0, F_i\}$ with $F_0(x) := x$ and $F_i(x) := (x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$. The new item selection x is accepted if it does not violate the weight limit, that is, $H(x) \le 0$ is fulfilled where $H(x) = \sum_{j=1}^{n} x_j \cdot w_j - W$.

To omit unpromising solutions we use the function $f \colon \mathcal{S} \to \mathbb{R}^2$ with

$$f(x) = \left(\sum_{i=1}^{n} x_i \cdot p_i, \sum_{i=1}^{n} x_i \cdot w_i \right)$$

and the partial order $\succeq_{\mathrm{par}}$ with $(p_1', w') \succeq_{\mathrm{par}} (p_1, w)$ iff $p_i' \ge p_i$ and $w' \le w$. The worst-case runtime of the explained DP is $O(n \cdot W)$ since $\sum_{i=1}^{n} |\mathcal{T}_{i-1}| \le n \cdot (W+1)$, that is, we store on the i-th level for each weight between 0 and W at most one set of at most i items.

The expected optimization time of the EA for the multi-objective knapsack problem is $O\big(n^2 \cdot W \cdot \log(n \cdot W)\big)$ due to Theorem 1.

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 known evolutionary algorithms proposed in the literature and allows to treat other problems such as the knapsack problem as well.

6. REFERENCES

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

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

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

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

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

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

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

[8] 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.

[9] C. Horoba. Analysis of a simple evolutionary algorithm for the multiobjective shortest path problem. In *Foundations of Genetic Algorithms Workshop (FOGA)*. ACM Press, 2009. To appear.

[10] D. I. Kogan. Dynamic programming and discrete multicriterial optimization, 2004. In Russian.

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

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

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

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

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

[16] 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.

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

[18] 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.

[19] M. Theile. Exact solutions to the traveling salesperson problem by a population-based evolutionary algorithm. In *European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)*. Springer, 2009. To appear. Available at `http://www.math.tu-berlin.de/coga/people/~theile/publications/tsp.pdf`.

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

[21] 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.