Parameterized Maximization

Henning Fernau

WSI-2001-22

Henning Fernau
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
Germany

E-Mail: `fernau@informatik.uni-tuebingen.de`
Telefon: (07071) 29-77565
Telefax: (07071) 29-5061

Parameterized Maximization

Henning Fernau
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
email: `fernau@informatik.uni-tuebingen.de`

December 18, 2001

Abstract

Maximization problems are usually considered as parameterized problems taking as parameter the entity which has to be maximized. In contrast, we study here another parameterization, given by the dual bounding constant in the case of an integer linear program formulation. This way, it makes sense to consider parameterized maximization problems such that it can be well motivated to expect the parameter to be small. We give examples of fixed parameter tractable maximization problems, as well as of problems which are not expected to be fixed parameter tractable unless $FPT = W[1]$, a hypothesis which is generally considered unlikely in parameterized complexity.

1 Introduction

Optimization problems offer a rich source of problems which tend to have no polynomial-time algorithms, unless $P = NP$. They can be easily converted into decision problems by asking, given an integer k, whether the value of the given instance of the optimization problem lies above the threshold k (in the case of maximization problems) or below k (in the case of minimization problems). This is also the usual way of producing so-called parameterized problems out of optimization problems. Then, the threshold k (defined as above) is considered as the parameter of the corresponding parameterized problem. This strategy is elaborated in [4, 6]. The idea of the "good news section" of parameterized complexity is to design algorithms running in time $f(k)p(n)$, where k is the parameter of the problem, f is some arbitrary function, n is the overall size of the problem instance and p is some polynomial. The corresponding class of problems is usually denoted by FPT. On the other hand, there is also a notion of parameterized hardness, formalized best by the notion of $W[1]$-hardness, corresponding to the notion of NP-hardness in classical complexity theory. We will not define this notion formally here, but refer the interested reader to the monograph [6]. The generic $W[1]$-hard problem is the question whether a given nondeterministic Turing machine halts in k steps or not, where k is the parameter. Therefore, the same intuition backs the hypotheses $P \neq NP$ and $FPT \neq W[1]$: There seems to be no general method for simulating n steps of a nondeterministic Turing machine by a deterministic Turing machine running only $p(n)$ steps, where p is some polynomial.

The modified concept of feasibility—formalized by the class FPT—was developed in order to claim the polynomial solvability of the problem under the assumption that k is small (and independent of n). In actual fact, it sounds quite reasonable to assume that k is small when k is derived as the threshold of a typical minimization problem. In contrast, it looks a bit queer to assume that k is small when k is the threshold of a maximization problem. One way out would be to re-parameterize the problem by choosing $(n-k)$ instead of k as the parameter, when some natural upper bound n on k is known; e.g., in the case of maximum independent set, this would transfer the original problem basically into a minimum vertex cover problem. It was observed in various examples that this sort of re-parameterization usually changes the character of the problem drastically (when viewed as a parameterized problem). For example, the question, given a graph, whether there is an independent set of size (at least) k (where k is the parameter) is $W[1]$-hard, while the question of deciding whether a graph contains a vertex cover of size (at most) k lies in FPT.

By way of contrast, we study here another parameterization, given by the dual bounding constant in the case of an integer linear program (ILP) formulation. This way, it makes also perfect sense to consider parameterized maximization problems such that it can be well motivated to expect the parameter to be small. We give examples of fixed parameter tractable maximization problems, as well as of problems which are not expected to be fixed parameter tractable unless $FPT = W[1]$, a hypothesis which is generally considered unlikely in parameterized complexity.

The paper is organized as follows: In Section 2, we discuss the well-known 0-1 knapsack problem as an introductory example. In Section 3, we provide a definition of the central notion of this paper, i.e., (fixed parameter tractable) parameterized maximization problems. In Section 4, we will see more parameterized maximization problems which are fixed parameter tractable, whilst in Section 5, we will learn about parameterized maximization problems which do not allow for fixed parameter algorithms unless $FPT = W[1]$.

2 0-1 knapsack: a first example

One of the classical NP-complete problems is the following knapsack problem, formulated here as a maximization problem: We are given a set X of n items, i.e., $X = \{x_1, \dots, x_n\}$, where each item x_i has some size $s_i \in \mathbb{N}$ and some profit $p_i \in \mathbb{N}$. We would like to pack our knapsack of maximal capacity $b \in \mathbb{N}$ such that the profit we might expect by somewhere carrying this knapsack is maximized.

More formally, the problem can be formalized by using the following ILP formulation:

$$\max \sum_{i=1}^{n} p_i x_i$$

subject to

$$\begin{array}{rcl} \sum_{i=1}^{n} s_i x_i & \leq & b \\ (x_1, \dots, x_n) & \in & \{0,1\}^n \end{array}$$

So, the items x_i are here interpreted as boolean variables.

The knapsack problem given above has several peculiarities; in particular, it has a polynomial-time algorithm when assuming that the numbers involved in the specification of a problem instance are given in unary. Therefore, we will assume in the following that *all numbers are encoded in binary.*

The usual transformation of this maximization problem would yield the following parameterized problem:

Input: n items with sizes s_i and profits p_i, the knapsack capacity b, and the profit threshold k.

Parameter: k

Question: Is there a subset of items which yield a profit larger than k and has an overall size of less than b?

Is this parameterization chosen in accordance with the nature of the problem? I think this is not the case. Anyone posing this problem will be happy to obtain a solution having a profit as large as possible, so the basic assumption underlying the development of fixed parameter algorithms is violated if the parameter is chosen to be the threshold value of the profit (which has to be maximized).

So, it appears to be more reasonable to take the knapsack capacity b as the parameter of the problem (whose parameterized formulation is identical, otherwise). This means that we are interested in large profits while carrying only a backpack of reasonably small weight or size. This seems to be a comprehensible assumption from the practical point of view. What is even more intriguing is the fact that one could now afford keeping the original character of the problem (as a maximization problem). This way, we get the following parameterized maximization problem:

Input: n items with sizes s_i and profits p_i, and the knapsack capacity b.

Parameter: b

Question: What is the maximum profit obtainable by choosing a subset of items which has an overall size of less than b?

The fact that this latter problem is fixed parameter tractable can be seen by employing two different strategies, as elaborated in the following. Although the ideas behind the strategies are known from the literature, the way we use them is somehow special.

Reduction to problem kernel: First of all, one can get rid of all items whose size is larger than b, because they will never belong to any feasible solution. Then, there can be at most one item of size b, one item of size $b-1, \ldots,$ and at most one item of size $\lceil (b+1)/2 \rceil$. Moreover, there can be at most two items of size $\lfloor b/2 \rfloor, \ldots,$ up to size $\lceil (b+2)/3 \rceil$, etc. Altogether, it is not reasonable to have more than

$$b/2 \cdot 1 + b/6 \cdot 2 + b/12 \cdot 3 + b/20 \cdot 4 + \ldots + 1 \cdot b \in O(b \log b)$$

many items in the problem instance; more precisely, e.g., if there are more than two items of size $\lfloor b/2 \rfloor$, then we can safely omit all items of this size of the original problem instance but the two ones yielding the largest and second largest profit of all items of this size.

Note that this problem kernel reduction does not yield a "problem kernel" whose size (measured in the number of items) is polynomial in the size of the parameter, but which is exponential, since all numbers are encoded in binary. After having sorted the original instance according to decreasing size, braking ties by sorting according to decreasing profit, it is easy to implement the kernelization to run in time $O(n \log n)$ (disregarding the size of the numbers).

Taking into account that a knapsack problem is specified by n items and the profit and size of each item (and the knapsack capacity b itself), one easily sees that the size of a "problem kernel" is bounded by $(b \log b) * (p_{max} + s_{max}) + b$, where p_{max} and s_{max} denote the maximum profits and sizes (among the p_i and s_i, respectively). Since trivially $s_{max} \leq b$, we have now a "kernel" of size $f(b) p_{max}$ for some suitable function f. Note that usually a problem kernel (used for deriving parameterized algorithms) is only allowed to depend on the parameter and not on other parts of the instance.

Nevertheless, by cycling through all $2^{O(b \log b)}$ possibilities of taking or not taking items into the knapsack, we can easily arrive at an FPT-algorithm for solving the 0-1 knapsack problem.

Dynamic programming: It is well-known that knapsack admits a pseudo-polynomial time algorithm.[1] This means that there is an algorithm running in time $O(n \sum_{i=1}^{n} p_i)$ which solves the 0-1 knapsack problem. This algorithm is based on dynamic programming. For our needs, we employ a related dynamic programming strategy leading to an algorithm running in time $O(nb \log(p_{max}))$, which is again a pseudo-polynomial time algorithm for the dual knapsack minimization problem (see [8]). .

One constructs a table $T[\cdot,\cdot]$ with $T[a,j]$ (with $1 \leq a \leq b$ and $1 \leq j \leq n$) containing the maximum profit of solving the given knapsack instance, but only putting (some) of the first j items into the knapsack (which now has maximum capacity of a). Disregarding trivial initializations, we can compute

$$T[a,j] = \max\{T[a,j-1], T[a-s_j, j-1] + p_j\}.$$

The entries of the tables are each of size p_{max} at most, which gives the claimed running time, taking into account the binary encoding of p_{max}.

Hence, we have the following result:

Theorem 2.1 Knapsack maximization (with n items and knapsack capacity bounded by b) can be solved in time $O(\log(p_{max})(n \log n + b^2 \log b))$. $\square$

Let us remark that the parameterized tractability of knapsack maximization can be also shown in a third way: The usual way to derive the fully polynomial time approximation scheme (FPTAS) for knapsack can be modified so that it yields also an FPTAS of the dual problem (in the sense of ILP), see [8]. According to the argument proving that every problem admitting an FPTAS is fixed parameter tractable [4, Theorem 3.2], one can prove that knapsack maximization is also fixed parameter tractable. In [4], it was shown in this way that the parameterized knapsack problem is fixed parameter tractable when taking the obtainable profit threshold as parameter. Similarly, the somewhat more general notion of efficient polynomial time approximation scheme (introduced in [5]) can be exploited for designing parameterized algorithms for maximization problems.

We observe that in the case of the maximum subset sum problem (which is just the 0-1 knapsack problem with the restriction that, for all i, $p_i = s_i$), the size of the "problem kernel" derived above only depends on b (since, of course, $p_{max} = s_{max} \leq b$). Hence, we get an algorithm running in time $O(n(\log n)(\log b) + (b \log b)^2)$ in this case.

[1]More details can be found in [3, 8, 9].

3 Parameterized maximization: formalizing notions

Now that we have discussed one example of a parameterized maximization problem, it is time to define more formally what we mean by the term "parameterized maximization." We try to keep this notion close to the one used for (non-parameterized) optimization problems as included in [3].

A *parameterized maximization problem* $\mathcal{P}$ can be specified by

1. the set of instances $I_{\mathcal{P}} \subseteq \{0,1\}^*$,
2. for each instance $x \in I_{\mathcal{P}}$, the set of feasible solutions $S_{\mathcal{P}}(x)$,
3. a measure function m yielding the value $m(x,y)$ for any feasible solution $y \in S_{\mathcal{P}}(x)$,
4. a distinguished part of the input instance x called the parameter (of size) k; k is considered to be independent of $|x|$.

For technical reasons, we pose some further requirements:

1. $I_{\mathcal{P}}$ and, for each $x \in I_{\mathcal{P}}$, also $S_{\mathcal{P}}(x)$ should be decidable in fixed parameter polynomial time, the parameter being again k.
2. Moreover, there is a polynomial q and a function f such that

$$\forall x \in I\ \forall y \in S_{\mathcal{P}}(x) : |y| \leq f(k)q(|x|)$$

   and $\forall y \in \{0,1\}^*$ with $|y| \leq f(k)q(|x|)$, the question $y \overset{?}{\in} S_{\mathcal{P}}(x)$ is decidable in fixed parameter polynomial time.

3. Finally, m should be computable in fixed parameter polynomial time, where k is the parameter.

We call a parameterized maximization problem $\mathcal{P}$ *fixed parameter tractable*, or $\mathcal{P} \in FPT-\max$, if there is some (deterministic) algorithm $\mathcal{A}$ which, given some $x \in I_{\mathcal{P}}$, returns

$$m^*(x) = \max\{m(x,y) \mid y \in S_{\mathcal{P}}(x)\}$$

if $S_{\mathcal{P}}(x) \neq \emptyset$; otherwise, the algorithm may loop forever. The running time of $\mathcal{A}$ should be bounded by $f(k) \cdot p(|x|)$, where f is some arbitrary function and p is some polynomial.

Obviously, the 0-1 maximum knapsack problem considered in the previous section is fixed parameter tractable in this more formal way. We get to know more fixed parameter tractable maximization problems in the next section.

Note: Instead of requiring that the running time of the maximization algorithm $\mathcal{A}$ should be bounded by $f(k) \cdot p(|x|)$, one could equivalently require that the running time is bounded by $f(k) + p(|x|)$. This fact can be proved as in the case of the class of decision problems FPT, see [6].

4 Other maximization problems which are fixed parameter tractable

One argument for showing that knapsack maximization if fixed parameter tractable was based on a pseudo-polynomial time algorithm for the dual knapsack problem. A similar argument applies to other knapsack-like problems where also a pseudo-polynomial time algorithm is known, see [3, 8, 9].

The kernelization argument also applies to the general assignment problem, where we have to fill not only one but m knapsacks of different capacities $b_1, \ldots, b_m$. The parameter would then be the sum of the b_is. In a similar but less efficient fashion we can treat variants of the knapsack problem where the items are not 1-dimensional but 2- or 3-dimensional objects. Many other variants of knapsack can be treated in this way, too.

Interestingly, I. Aho [1, 2] formulated a graph problem (which can easily seen to be a generalization of the 0-1 knapsack problem) which can also be shown to belong to the class of fixed parameter tractable maximization problems; the *maximum weight-constraint path problem, MWCP*, is given as follows:

Input: A directed acyclic graph $G = (V, E)$, where each edge e has a length $l(e)$ and a weight $w(e)$, specified vertices $s, t \in V$ and a weight constraint W.
[All numbers are natural numbers encoded in binary.]

Parameter: W

Question: What is the maximum length of a (simple) path in G from s to t with total weight of W or less?

Theorem 4.1 MWCP is fixed parameter tractable.

Proof. The assertion can be seen by using dynamic programming: For each triple (v, v', w) with $v, v' \in V$ and $1 \leq w \leq W$, put into a table T the maximum length of a path leading from v to v', whose total weight equals w (if it exists). The running time is obviously a polynomial in $|V|$ and in the number W. □

The same algorithm shows, in addition to the NP-completeness shown by Aho [2], that MWCP admits a pseudo-polynomial time algorithm.

As a concluding example, observe that typical minimization problems like vertex cover can be easily and meaningfully enriched by additional maximization constraints, which then would lead to a sort of parameterized maximization problem we could consider as in the previous examples. More precisely, let us consider the problem *vertex cover maximization VCM* given as follows:

Input: A graph $G = (V, E)$, profits (natural numbers) $p(v)$ for each vertex $v \in V$, a natural number k.

Parameter: k

Question: What is the maximum profit $p(C)$ of a vertex cover $C \subseteq V$ with $|C| \leq k$?[2]

A motivation for considering this problem is the following: The usual covering problem instance might correspond to the establishment of a service center net which should cover all main road connections (modelled by edges). Although the management, for reasons of limited investment, yielded only the installment of k such centers, each of the (potential) centers would have a peculiar expected profit, e.g., depending on the size and structure of the local population, and the overall profit should be maximized.

Theorem 4.2 VCM is a fixed parameter tractable maximization problem.

Proof. We consider the following variant of Buss' kernelization of vertex cover.

1. Initialize $\tilde{C} := \emptyset$.
   Use the following two kernelization rules as long as possible:

   - If v is a vertex with no neighbours, v can be removed from the graph, since v will not be part of any <u>minimal</u> vertex cover.
   - If v is a vertex of degree greater than k, v must be in any vertex cover, since otherwise all neighbours would be in the cover, which is not feasible since we are looking for vertex covers with at most k vertices. Hence, we can remove v (and its incident edges) from the graph. Moreover, add v to $\tilde{C}$.

   After having applied these kernelization rules exhaustively, we are left with a graph G' with at most k^2 vertices.

2. As a further preparatory step, we sort the vertices of the originally given graph according to decreasing profit.

3. $max := 0$;

4. `For each` of the minimal vertex covers C in G' (which can be searched for exhaustively)[3] `do`:

[2]Recall that a vertex set C is called *vertex cover* if each edge from E is incident to at least one vertex from C.

[3]In fact, a search-tree technique shows that there are no more than 2^k many minimal vertex covers in G', see [7].

```
   (a) If $|C \cup \tilde{C}| < k$ then add the most profitable $k - |C \cup \tilde{C}|$ vertices
       of $G$ (which are not already included in $C \cup \tilde{C}$) to $C$.
   (b) If $p(C \cup \tilde{C}) > max$ then $max := p(C \cup \tilde{C})$.
5. Return $max$.
```

Obviously, the running time of this algorithm is $O(n(\log(n) + k) + f(k))$ for some function f. □

Alternatively, one could prove the statement of the previous theorem by giving a search-tree algorithm of the following form:

```
search-tree($G = (V, E), G' = (V', E'), k, C, p$)
1. If $k = 0$ and $E' = \emptyset$ then return $C$
2. else if $k > 0$ and $E' \neq \emptyset$ then
3. (a) Choose some $e = \{v_1, v_2\} \in E$.
   (b) $C_1$ :=search-tree($G, G' - v_1, k - 1, C \cup \{v_1\}, p$);
   (c) $C_2$ :=search-tree($G, G' - v_2, k - 1, C \cup \{v_2\}, p$);
   (d) If $p(C_1) > p(C_2)$ then return $C_1$ else return $C_2$.
4. else if $k > 0$ and $E = \emptyset$
5. then let $C'$ contain the $k$ most profitable vertices from $V \setminus C$;
       return $C \cup C'$
6. else if $k = 0$ and $E \neq \emptyset$ then return $\emptyset$.
```

Since the empty set has the lowest possible profit, namely zero, we could afford returning the empty set indicating that there was no valid vertex cover found at the corresponding search path. In order to obtain the desired result, we call the procedure as follows:

```
1. $C$ :=search-tree($G, G, k, \emptyset, p$).
2. If $C \neq \emptyset$ then return $p(C)$.
```

Theorem 4.3 VCM $\in FPT - \max$. □

5 Hard maximization problems

In the previous two sections, we presented several parameterized maximization problems which are fixed parameter tractable. Here, we will show that there are also parameterized maximization problems which are at least as hard as $W[1]$-hard parameterized decision problems, so that these problems—under the assumption that $W[1]$-hard decision problems are not fixed parameter tractable—do not admit efficient parameterized algorithms.

Consider the following variant of the independent set problem, called *maximum weight-constrained independent set problem MWCIS*:

Input: A graph $G = (V, E)$, weights (natural numbers) $w(v)$ for each vertex $v \in V$, a natural number W.

Parameter: W

Question: What is the size of a maximum independent set $I \subseteq V$ with $w(I) \leq W$?

Theorem 5.1 If $FPT \neq W[1]$, then MWCIS $\notin FPT - \max$.

Proof. According to [6], the (usual) independent set problem on general graphs, where the parameter is an upper bound on the size of the wanted large independent set, is $W[1]$-hard. Consider some graph $G = (V, E)$. If MWCIS were fixed parameter tractable, then surely also the simplified version with $w(v) = 1$ for all $v \in V$ would be, too. Hence, if this restricted MWCIS version with parameter $W = k$ would return k as maximum size of some solution, then the an algorithm for the usually parameterized independent set problem could return YES, whilst it should return NO if the MWCIS algorithm returns a value less than k as maximum size of some solution. □

Similarly, the non-existence of a fixed parameter algorithm for the analogously defined *maximum weight-constrained clique problem MWCC* can be backed:

Corollary 5.2 If $FPT \neq W[1]$, then MWCCS $\notin FPT - \max$. □

The two problems considered up to this point in this section had a somewhat artificial flavour. Let us therefore investigate now two maximization problems (found in the literature) which already possess a natural parameter in their original formulation.

The *maximum edge subgraph problem MES* (see [3, GT35]) is defined as follows:

Input: A graph $G = (V, E)$, weights (natural numbers) $w(e)$ for each edge $e \in E$, a natural number k.

Parameter: k

Question: What is the maximum weight of the set of edges in a subgraph induced by some vertex set $V' \subseteq V$ with $|V'| = k$?

Theorem 5.3 If $FPT \neq W[1]$, then MES $\notin FPT - \max$.

Proof. According to [6], the clique problem is $W[1]$-hard, where the parameter is an upper bound on the size of the wanted large clique. Consider some graph $G = (V, E)$. If MES were fixed parameter tractable, then surely also the simplified version with $w(e) = 1$ for all $e \in E$ would be, too. MES on (G, w, k) would yield $k(k-1)/2$ iff G contains a clique of size k. $\Box$

The *maximum minimum metric k-spanning tree problem MMMST* (see [3, ND5]) is defined as follows:

Input: A graph $G = (V, E)$, lengths (natural numbers) $\ell(e)$ for each edge $e \in E$ satisfying the triangle inequality, a natural number k.

Parameter: k

Question: What is the maximum weight of the set of edges in a minimum spanning tree of a subgraph induced by some vertex set $V' \subseteq V$ with $|V'| = k$?

Theorem 5.4 If $FPT \neq W[1]$, then MMMST $\notin FPT - \max$.

Proof. We again reduce to the clique problem. Consider some graph $G = (V, E)$. If MMMST were fixed parameter tractable, then surely also the simplified version with $\ell(e) \in \{0, 1\}$ for all edges e would be, since the triangle equality would be satisfied. Consider the complete graph $G' = (V, E')$ with $\ell(e) = 1$ if $e \in E$ and $\ell(e) = 0$ if $e \notin E$. If G contains a k-clique, then MMMST on (G', ℓ, k) would yield $k - 1$ (by choosing a vertex set V' inducing a clique). If G contains no k-clique, MMST would yield at most $k - 2$. $\Box$

There is also a kind of generic parameterized maximization problem which is not fixed parameter tractable unless $FPT = W[1]$, namely *maximum profit short Turing machine computation MPSTM*:

Input: A nondeterministic Turing machine TM with profits $p(t)$ and weights (natural numbers) $w(t)$ for each transition t, a natural number W.

Parameter: W

Question: What is the maximum profit of any path in the computation graph of TM, corresponding to a terminating computation with weight less than W?

Theorem 5.5 If MPSTM were a fixed parameter tractable maximization problem, then $FPT = W[1]$.

Proof. As mentioned in the introduction, the problem of determining whether a given nondeterministic Turing machine TM halts within k steps is the generic $W[1]$-complete problem. So, take some nondeterministic Turing machine TM and modify it as follows:
(1) every transition of the original TM gets profit 2 and weight 1, and
(2) add a new state and transitions of profit 1 and weight 0 from every original state to this new state.
It is easy to see that an MPSTM algorithm, when given this new Turing machine and k as weight parameter, answers 1 if and only if there was no halting computation path of length of at most k of the original Turing machine TM. □

Actually, MPSTM could be used as generic problem to define a class of parameterized maximization problems $W[1] - \max$ corresponding to $W[1]$, as shown in the following. The following theorem shows that all problems considered in this section belong to $W[1] - \max$.

Theorem 5.6 MWCIS, MWCC, MES and MMMST are solvable with the help of MPSTM.

Proof. The graph (as instance of MWCIS) is coded within the transition relation of the Turing machine. A transition step of profit 1 and of weight w consists in choosing a vertex v of the graph with $w(v) = w$ to be included in the independent set I which is going to be constructed. A unique letter is reserved for each vertex, and each chosen vertex is written onto the (otherwise empty) tape. This can be done within k steps. Then, there are further $p(k)$ steps of zero weight and zero profit of the Turing machine which check whether there are no edges connecting the chosen vertices. Hence, in $p'(k)$ steps, the nondeterministic Turing machine (k of them being nondeterministic steps) will find an independent set of size k, if such a set exists.[4]

A similar argument applies for MWCC.

In the case of MES, after guessing a vertex subset V' in (at most) k steps each of weight 1 and of zero profit, the TM checks all connections of the graph induced by V' (these connections are stored in the finite memory of the TM): each checking step has zero weight; if an existing edge of weight w is checked, than the corresponding checking transition has profit w.

In the case of MMMST, the Turing machine first guesses a vertex subset V' in k steps each of weight 1 and of zero profit, then it computes a minimum spanning tree, and finally the weight of this tree is computed (using non-zero weights on the transitions). □

Note: MPSTM is quite similar to MWCP, in the sense that both problems

[4]Although this is of no importance in the general notion of fixed parameter (in)tractability employed here, note that the functions p and p' are polynomials of low degree.

look for maximum profit minimum weight paths in graphs. The difference is that the reachability graph of a Turing machine is compactly codified as a Turing machine, while for MWCP, the graph is explicitly given.

6 Conclusions

We discussed a new and, in our opinion, more natural parameterization of maximization problems and showed that several problems can be solved efficiently in the parameterized setting in this way. The corresponding algorithms were based on different techniques, namely:

- reduction to problem kernel,
- dynamic programming (in connection with problems admitting pseudo-polynomial time algorithms),
- directly using (variants of) FPTAS algorithms,
- search tree techniques, and
- enumeration algorithms.

Our focus here was rather on providing examples for the various algorithmic techniques than to chase for efficiency. Hence, the most favourite algorithmic techniques for designing FPT algorithms are also useful for developping parameterized algorithms for maximization problems. From this point of view, it is an interesting new observation that pseudo-polynomial time algorithms can be seen as fixed parameter algorithms.

On the other hand, we also showed examples of problems which do not allow for fixed parameter maximization algorithms unless $FPT = W[1]$. Therefore, our observations raise the natural question of further developping a theory of parameterized maximization problems.

Note that typical parameterized maximization problems have two different kinds of restrictions: hard ones (formalized by the parameter) and somewhat weaker ones (formalized by the entity to be optimized) in the sense that any solution which is going to be considered at all in the process of seeking an optimal solution has to firstly satisfy the hard restriction. In the introductory knapsack example, this meant that we are only interested in finding maximal solutions among those solutions which fit into our small backpack. Considering this two-stage process more abstractly, it would also make sense to ponder parameterized minimization problems, where, in a first stage, those feasible solutions are selected which guarantee a small parameter, and then a minimal solution is sought for among these feasible solutions.

As a concrete example, observe that R. Hassin developped a pseudo-polynomial time algorithm (and, based on this, a fully polynomial time approximation scheme) for the shortest weight-constrained path problem, see [3, ND43], which is basically the minimization variant of MWCP as discussed above. Further variants of this problem were studied in [10]. It would be interesting to see other natural examples for this sort of parameterized minimization problems.

Finally, it might be interesting to develop parameterized approximation algorithms for optimization problems which are probably not in $FPT - \max$.

Acknowledgments:

We are grateful for discussions with I. Aho, J Gramm and K.-J. Lange.

References

[1] I. Aho. Interactive knapsacks. *Fundamenta Informaticae*, 44:1–23, 2000.

[2] I. Aho. On the approximability of interactive knapsack problems. In L. Pacholski and P. Ružička, editors, *SOFSEM'01; Theory and Practice of Informatics*, LNCS, pages 152–159. Springer, 2001.

[3] G. Ausiello, P. Creczenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation; Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.

[4] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *Journal of Computer and System Sciences*, 54:465–474, 1997.

[5] M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64:165–171, 1997.

[6] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[7] H. Fernau. On parameterized enumeration. Technical Report WSI–2001–21, Universität Tübingen (Germany), Wilhelm-Schickard-Institut für Informatik, 2001.

[8] G. V. Gens and E. V. Levner. Computational complexity of approximation algorithms for combinatorial problems. In L. Bečvář, editor, *Mathematical Foundations of Computer Science MFCS*, volume 74 of *LNCS*, pages 292–300. Springer, 1979.

[9] S. Martello and P. Toth. *Knapsack Problems; Algorithms and Computer Implementations.* John Wiley & Sons, 1990.

[10] C. A. Phillips. The network inhibition problem. In *Proceedings of the 25th Annual Symposium on the Theory of Computing*, pages 776–785. ACM, 1993.