Parameterized Complexity - an Overview

Uffe Flarup[1]
flarup@imada.sdu.dk

[1]Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark

February 5, 2007

Outline

Introduction

Many important problems in computer science, engineering, mathematics are unfortunately *NP-hard* problems

Heuristics, parallel algorithms (branch'n'bound), approximation schemes, randomized algorithms are important aspects in order to solve many hard problems

Fixed Parameter Tractability is yet another approach to combat intractability of important problems. Originally introduced by Downey and Fellows

Heuristics

Good news: Good running time, (often) good solutions, uses techniques that makes sense for the given problem

Bad news: Often no knowledge about quality of solution (of course, the solution might be better than previously best known solution), few mathematical results

Parallel algorithms (b'n'b)

Good news: We get the exact/optimal solution, many combinatorial search problems easy to parallelize, computing power quite cheap

Bad news: Exponential running time (large cluster/botnet required), not applicable to really large instances

Approximation algorithms

Good news: Polynomial running time, knowledge about quality of solution

Bad news: We rarely get the optimal solution, many problems cannot be approximated very well (e.g. CLIQUE), some PTAS's are impractical

Fixed parameter tractable

Good news: We get the exact/optimal solution

Bad news: Worst case exponential running time... But, WAIT!

Outline

General idea

Fixed Parameter Tractability is a design paradigm where we take into account, that for many "real life" problems we know in advance, that some part of the instance will be small

With the assumption that a certain part of the instance (the parameter) is small, we want to develop efficient (polynomial time) algorithms

While a problem in general might be intractable, not all *instances* of that problem need to be intractable. Framework to analyse which instances are (in)tractable

Example: Database Queries

Consider evaluating a sentence of some language (a query) in a given finite structure (a database)

In *general* this is of high complexity, but typical queries are *much simpler* (delete * from students where average $<$ "D")

Thus, we should put more focus on simple queries in large databases, than on complex queries in general. If query complexity k is small and database size n is large, then $O(2^k \cdot n)$ is better than $O(n^k)$

Parameterized complexity provides a framework to study this

The parameter

Typical parameters are:

- Number of variables in a *SAT* instance
- Number of clauses in a *SAT* instance
- Treewidth/pathwidth/cliquewith of a graph
- Size of vertex cover/clique/dominating set of a graph

A problem can have multiple parameterizations, each leading to different results

Class *FPT*

A parameterized language L is a subset of $\Sigma^* \times \Sigma^*$, where Σ is a finite alphabet. Let $(x, k) \in L$, then we call x the *main part* and k the *parameter*. Often k is an integer

$L \in FPT$ if it can be decided in time $f(k) \cdot |x|^{O(1)}$ (or $f(k) + |x|^{O(1)}$) for an *arbitrary* function f (typically exponential at least)

For a fixed k it is in P, moreover for every k it is in the *same* polynomial class via the *same* machine

Outline

Example: VERTEX COVER

Parameter k is the size of the vertex cover we are looking for

Straight forward $O(n^k)$ approach does not lead to *FPT* algorithm

Instead we split the algorithm into two parts (this approach due to Sam Buss):

- Reduction of the graph to a *problem kernel* - a graph of size $O(k^2)$
- Brute force search to solve the reduced problem instance

Part 1: Kernelization

Repeat the following rules as many times as possible:

- If G has a vertex of degree $> k$, include it in cover
- If G has a vertex of degree 0, exclude it from cover

Let k' be k minus number of vertices included in this step
If we end up with a graph with more than $k' \cdot (k+1)$ vertices, then reject

Running time is $O(k \cdot n)$

Part 2: Search tree

Create a search tree of height at most k'

Choose any remaining vertex v and branch as follows:

- Include v in cover
- Exclude v from cover, but include all its neighbors

Explore exhaustively all paths in this search tree for a vertex cover of size k'. Eventually either accept or reject

Running time $O(2^k)$

Improved algorithm

Kernelization: Additional rules to remove vertices of degree 1, 2 or 3

Search tree: Use degree bounds to construct smaller tree. More involved than idea given here

Total running time becomes $O(k \cdot n + 1.286^k)$

Implementation shows it is *practical* for graphs of arbitrary size and $k \leq 400$

FPT roundup

Analysis of running time now done with respect to two variables: Size of input and size of parameter

Size of parameter contributes only as a multiplicative factor, *not* to the degree

Kernelization followed by brute force search is an important technique

It is *good news* that a problem can have more than one parameterization, however there is *bad news* as well...

Outline

Parametric intractability

Intractability is (as usual) shown by means of *hardness* for a certain complexity class

Consider the following classical problem:

NONDETERMINISTIC TURING MACHINE ACCEPTANCE

Input:	A nondeterministic Turing machine M
Question:	Does M have an accepting computation in $\leq |M|$ steps?

Conjecture: This problem is intractable. The behavior of a Turing machine in general is so complex, that we cannot decide if it accepts without inspecting all computation paths

Parametric intractability

Downey and Fellows conjectured that the equivalent *parameterized* version would be intractable also

SHORT NONDETERMINISTIC TURING MACHINE ACCEPTANCE

Input:	A nondeterministic Turing machine M
Parameter:	A positive integer k
Question:	Does M have an accepting computation in $\leq k$ steps?

Straight forward $O(|M|^k)$ approach does not lead to FPT algorithm

Note: If the number of states in M is bounded by a constant, the problem is in FPT

Reductions

A parameterized reduction is a transformation from one parameterized language L to another parameterized language L'. We require the following to be satisfied:

- $(x, k) \in L$ iff $(x', k') \in L'$
- $k' = g(k)$, depends *only* on the parameter and *not* on the main part
- x' can be computed in time $f(k) \cdot |x|^{O(1)}$
- Functions f and g only depends on the problems

As usual, if L reduces to L' and $L' \in FPT$, then $L \in FPT$ as well

Note: Karp reductions are rarely parameterized reductions (the reduction between CLIQUE and INDEPENDENT SET being an exception)

The $W[t]$-hierarchy

$$FPT \subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \dots$$

The $W[t]$-hierarchy is based on a modified CIRCUIT SATISFIABILITY problem:

- *Small gate:* Constant fan-in
- *Large gate:* Arbitrary fan-in, depending on the instance
- *Depth:* Max number of gates on an input-output path
- *Weft:* Max number of *large* gates on an input-output path

The $W[t]$-hierarchy

$W[1]$: The class of languages that can be reduced to a family of constant depth, *weft 1* circuits, such that the produced circuit has a *weight* k' satisfying assignment iff the original instance satifies $(x, k) \in L$

For all classes in the $W[t]$-hierarchy the *parameter* is always the same: The *weight* of the satisfying assignment

The difference between the classes in the $W[t]$-hierarchy is the *weft* of the circuit

Ex.: INDEPENDENT SET

Hardness for $W[1]$ is an involved proof (not shown here). Instead membership in $W[1]$ will be illustrated:

- Each vertex in G correspons to one input gate in the circuit
- For every edge (v, u) in G we build a *small* OR-gate of constant fan-in 2: $(\neg v \vee \neg u)$
- Output from small gates are given as input to a single *large* AND-gate

G has an independent set of size k iff this circuit has a satisfying assignment of weight k

Unlike NP-hardness results, $W[1]$-hardness for SHORT TURING MACHINE ACCEPTANCE uses INDEPENDENT SET and CLIQUE as intermediate results

Note: DOMINATING SET is $W[2]$-complete

Outline

Classical complexity

Some connections to classical complexity theory:

$FPT \neq W[1]$ implies $P \neq NP$
$FPT = W[1]$ implies $3SAT$ solvable in time $2^{o(n)}$

Not known if $FPT = W[1]$ implies anything about the rest of the $W[t]$-hierarchy

Practical intractability of problems in NP, which are unlikely to be complete for NP, can be shown using $W[t]$-hardness results

Algebraic complexity

The class *FPT* can easily be transferred to the Blum-Shub-Smale (BSS) model of computation over the real numbers

However, more than one natural way to consider a possible $W[t]_{\mathbb{R}}$-hierarchy, depending on parameter:

- Number of variables: If only equalities are allowed we can square and sum to get a single polynomial, and thus in $FPT_{\mathbb{R}}$ due to Renegar. If *in*equalities are allowed it seems intractable
- Number of non-zero values in a satisfying assignment: Seems intractable

Algebraic complexity

Different hierarchies can be constructed depending on whether large gates can be algebraic or only boolean

In the discrete setting, parameterizing by the number of variables in a SAT instance leads to an FPT result. In the real number setting, parameterizing by the number of variables in a polynomial system may not give an $FPT_{\mathbb{R}}$ result

In the BSS model nondeterminism was introduced using certificate checkers. Parameterizing by number of steps gives an $FPT_{\mathbb{R}}$ result

Merci!