Fixed Parameter Algorithms and their Applications to CP and SAT

Barry O'Sullivan and Igor Razgon

Cork Constraint Computation Centre, University College Cork, Ireland

CP 2009 Tutorial

Details

Acknowledgements

Supported by Science Foundation Ireland Grant 05/IN/I886.

science foundation ireland
fondúireacht eolaíochta éireann

Where can I find these slides?

`http://www.cs.ucc.ie/~osullb/cp-tutorial-2009/`

Outline

The main purpose of this tutorial

Motivation

To supply you with basic skills that would allow you to start doing research in the area of fixed-parameter algorithms.

Relevant books

Recommended for the beginner

R. Niedermeier, “Invitation to Fixed-Parameter Algorithms”, Oxford University Press, 2006.

Others

- R. Downey and M. Fellows, “Parameterized Complexity”, Springer-Verlag, 1997.
- J. Flum and M. Grohe, “Parameterized Complexity Theory”, Springer, 2006.

Relevant books

Recommended for the beginner

R. Niedermeier, “Invitation to Fixed-Parameter Algorithms”, Oxford University Press, 2006.

Others

- R. Downey and M. Fellows, “Parameterized Complexity”, Springer-Verlag, 1997.
- J. Flum and M. Grohe, “Parameterized Complexity Theory”, Springer, 2006.

Outline

Introduction to Fixed Parameter Algorithms

The traditional view

The running time of an algorithm that solves an NP-Hard problem is exponential in the input size n, e.g. we believe that unless P=NP the running time of a SAT algorithm is $O(2^n)$.

The fixed parameter algorithm view (informal)

For some NP-Hard problems the running time of an algorithm is exponential in a parameter k, independent of n, and only polynomially depedent on n, e.g. vertex cover can be solved in $O(1.3^k + n)$, where k is the maximum number of vertices incident to all edges in the given graph.

Introduction to Fixed Parameter Algorithms

The traditional view

The running time of an algorithm that solves an NP-Hard problem is exponential in the input size n, e.g. we believe that unless P=NP the running time of a SAT algorithm is $O(2^n)$.

The fixed parameter algorithm view (informal)

For some NP-Hard problems the running time of an algorithm is exponential in a parameter k, independent of n, and only polynomially depedent on n, e.g. vertex cover can be solved in $O(1.3^k + n)$, where k is the maximum number of vertices incident to all edges in the given graph.

Why do we care about parameterised algorithms?

- For problems in which the parameter k is small we can find optimal solutions in time that is polynomial in n.
- We get provable upper bounds on the computation complexity of the problem.
- Exponential complexity is dependent only on k.

Why do we care about parameterised algorithms?

Traditional approaches to coping with NP-hardness include:

Exact methods, e.g. Branch-and-Bound

Advantage: precise
Disadvantage: non-scalable

Approximate methods, e.g. local search

Advantage: scalable
Disadvantage: imprecise

Fixed Parameter Methods

Fixed-parameter algorithms are scalable in the input size as well as precise.

What is the price for these advantages?

- Fixed-parameter algorithm takes an exponential time in terms of a parameter associated with a problem.
- If the parameter is small, this exponent can be considered as multiplicative or additive constant for a low-polynomial algorithm.

Example: CNF-Satisfiability

Example

Consider a boolean formula F in CNF over n variables and m clauses:

$$(x_1 \vee x_2) \wedge (\neg x_2 \wedge x_3 \wedge \neg x_4) \wedge \dots$$

The Fixed Parameter Algorithm Viewpoint

Parameter	Complexity
clause size	2-CNF is poly, 3-CNF is NP-complete
number of variables	2^n
number of clauses	1.24^m
formula length	1.08^λ where λ is total length

The total length of a formula is the number if literal occurrences in the formula.

Example: CNF-Satisfiability

Example

Consider a boolean formula F in CNF over n variables and m clauses:

$$(x_1 \vee x_2) \wedge (\neg x_2 \wedge x_3 \wedge \neg x_4) \wedge \ldots$$

The Fixed Parameter Algorithm Viewpoint

Parameter	Complexity
clause size	2-CNF is poly, 3-CNF is NP-complete
number of variables	2^n
number of clauses	1.24^m
formula length	1.08^λ where λ is total length

The total length of a formula is the number if literal occurrences in the formula.

Outline

Why should CP researchers know about this?

Less important reason

Fixed-parameter algorithms allow to efficiently solve a number of classes of CSP and SAT.

Most important reason

Our training and our skills as CP researchers ideally fit for doing research in the area of fixed-parameter algorithms which is full of very challenging open problems!

It's all about reformulation and preprocessing!

Designing fixed parameter algorithms can be regarded as an application for a formal approach to problem reformulation.

Cyclic cutset (a.k.a. feedback vertex set)

Problem statement

Input: a CSP Z, over *n* variables
Parameter: k, the cycle-cutset size.
Question: is it possible to remove at most k variables from Z so that the resulting CSP is acylic.

What do we know?

The problem is FPT and can be solved in time $O(5^k k^2 + poly(n))$.

Computing Backdoors

- A backdoor is a subset of variables of CSP (or SAT) whose deletion makes the resulting problem polynomially solvable.
- Given a backdoor, the instance can be solved by exploring all possible assignment to the backdoor variables and efficient solving the resulting residual instances
- It makes sense to use backdoors only if they are small.
- If computing a backdoor is NP-hard, this can be done by a fixed-parameter algorithm parameterized by the size of the backdoor.

Generic problem of backdoor computation

Problem statement

Input: an instance Z of CSP of SAT, a polynomially solvable class P of the given problem.
Parameter: k
Question: is it possible to remove at most k variables from Z so that the resulting instance belongs to P?

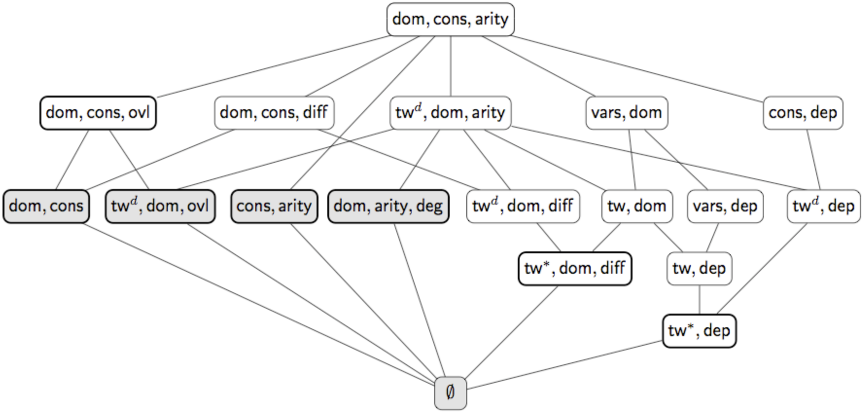
What do we know?

Some classes of this problem are FPT.

[Razgon and O'Sullivan, Journal of Computer and System Sciences, 2009.]

Parameterised Constraint Satisfaction

There are many ways to parameterise constraint satisfaction.



[Samer and Szeider, Journal of Computer and System Sciences, 2008.]

Global Constraints and Fixed-Parameter Algorithms

NVALUE Constraint

Enforcing domain consistent on NVALUE$([X_1, \ldots, X_n], N)$ is fixed parameter tractable in $k = |\bigcup_{i \in 1 \ldots n} dom(X_i)|$, but is $W[2]$-hard in $k = max(dom(N))$.

DISJOINT Constraint

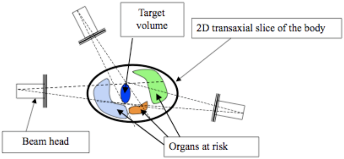
Enforcing domain consistent on DISJOINT$([X_1, \ldots, X_n], [Y_1, \ldots, Y_m])$ is fixed parameter tractable in $k = |\bigcup_{i \in 1 \ldots n} dom(X_i) \cap \bigcup_{j \in 1 \ldots n} dom(Y_j)|$,

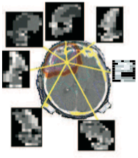
ROOTS Constraint

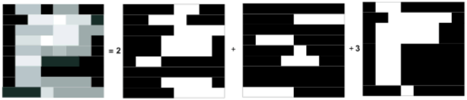
Enforcing domain consistent on ROOTS$([X_1, \ldots, X_n], S, T)$ is fixed parameter tractable in $k = |ub(T)\ lb(T)$.

[Bessiere et al., AAAI, 2008.]

Radiotherapy Treatment Planning







$$\begin{bmatrix} 0&3&3&0&2&2&0&0\\ 0&0&5&5&6&4&4&1\\ 0&3&3&3&5&5&2&2\\ 0&4&4&6&5&5&2&0\\ 0&3&3&2&3&2&2&0\\ 0&5&5&1&1&1&0&0\\ 0&3&3&0&0&0&0&0\\ 0&3&3&2&2&2&2&0\\ 1&1&1&4&2&2&2&2 \end{bmatrix} = 2 \begin{bmatrix} 0&0&0&0&1&1&0&0\\ 0&0&1&1&1&0&0&0\\ 0&0&0&0&1&1&1&1\\ 0&0&0&1&1&1&1&0\\ 0&0&0&1&1&1&1&0\\ 0&1&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&1&1&1&1&0\\ 0&0&0&0&1&1&1&1 \end{bmatrix} + \begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&1\\ 0&0&0&0&0&0&0&0\\ 0&1&1&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 1&1&1&1&0&0&0&0 \end{bmatrix} + 3 \begin{bmatrix} 0&1&1&0&0&0&0&0\\ 0&0&1&1&1&1&1&0\\ 0&1&1&1&1&1&0&0\\ 0&1&1&1&1&1&0&0\\ 0&1&1&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0 \end{bmatrix}$$

An FPT result in this domain enables CP to solve clinical sized instances to optimality.

[Cambazard, O'Mahony and O'Sullivan, CPAIOR, 2009.]

Outline

Definition of a fixed-parameter algorithm

Definition

A fixed-parameter algorithm is solves a problem in time $O(f(k) \times n^c)$, where:

- n is the input size
- k is the small parameter
- c is a constant independent on k
- $f(k)$ is an exponential function of k

Two important remarks to the definition

1. An algorithm with runtime $O(n^k)$ is not a fixed-parameter algorithm.
2. There may be two or more parameters. If the parameters are k and l, the runtime is $O(f(k, l) \times n^c)$.

Fixed-parameter tractable vs. intractable problems

1. If a problem can be solved by a fixed-parameter algorithm, it is called FIXED-PARAMETER TRACTABLE (FPT).
2. There are some problems that are not FPT unless some widely believed conjecture in complexity theory fails.

Fixed-parameter tractable vs. intractable problems

Classification

The existence of FPT and non-FPT problems raises the question of classification of a given problem into one of the classes. For some problems, the classification is a very challenging open question.

Two approaches to showing that our problem is FPT

1. Design a branch-and-bound based algorithm with the size of the search tree exponentially depending on the parameter, not on the input size.
2. At the preprocessing stage, transform the given instance into an equivalent one whose size depends on the parameter only.

Fixed-parameter tractable vs. intractable problems

Classification

The existence of FPT and non-FPT problems raises the question of classification of a given problem into one of the classes. For some problems, the classification is a very challenging open question.

Two approaches to showing that our problem is FPT

1. Design a branch-and-bound based algorithm with the size of the search tree exponentially depending on the parameter, not on the input size.
2. At the preprocessing stage, transform the given instance into an equivalent one whose size depends on the parameter only.

Fixed-parameter tractable vs. intractable problems

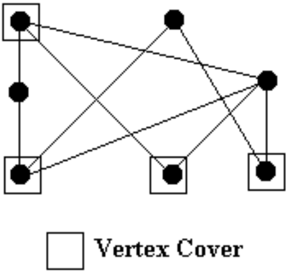
Observation for the CP Community

Both approaches are well known in the area of Constraint Programming, hence CP researchers are best trained to tackle hard problems related to fixed-parameter algorithms.

Example: Vertex Cover Problem

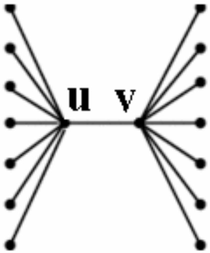
Vertex Cover Problem (VC)

Input: graph G of n vertices
Parameter: k
Question: is there a set of at most k vertices incident to all the edges of G



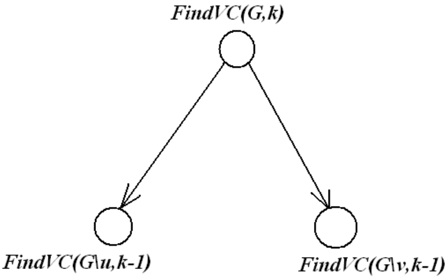
Fixed-parameter algorithm for Vertex Cover

```
FindVC(G,k)
    If G has no edges
       then return YES

    If k=0
       then return NO

    Select an edge {u,v} of

    If ( FindVC(G\u,k-1)
         or FindVC(G\v,k-1) )
       then return YES
       else return NO
```



Vertex Cover Runtime analysis

- The recursive applications of FindVC can be organized into a search tree.



- The height of the tree is at most k. Each non-leaf node has 2 children. Hence the search tree has $O(2^k)$ nodes.
- The complexity of `FindVC` is $O(2^k n)$.
- We have shown that the VC problem is FPT

Outline

Bounded search tree method

- The algorithm for VC problem is based on the methodology of bounded search tree.
- Such algorithms produce a search tree with:
  - a constant number of branches at each node
  - the height of each path from the root to a leaf depends on k
- The VC problem is a lucky case where the parameter can be reduced on each branch of the search tree. This allows us to easily control the height.

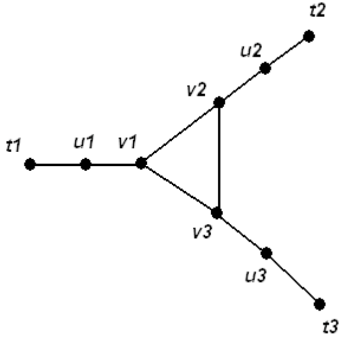
Bounded search tree method

- For a typical selection a subset of vertices having the given property there are two branches.
  - A vertex is selected (the good branch, the parameter is decreased).
  - A vertex is discarded (the bad branch, the parameter is not decreased).
- The resulting search tree has $O(n^k)$ nodes.
- More sophisticated techniques are required to cut the long paths caused by the bad branches.

Another Example: Multiway cut

Multiway cut

Input: graph G with specified vertices $t_1, \ldots, t_m$ called the terminals
Parameter: k
Question: is it possible to remove at most k non-terminal vertices to mutually separate all the terminals?



{v1,v2} is a multicut of size 2

NP-hard for $m > 2$

The branching structure

The branching structure

- Fix a terminal t_1 which is not separated yet from the rest of terminals
- Pick a vertex v adjacent to t_1
- On the first branch:
  - Remove v from the graph
  - Apply recursively to the resulting graph with the decreased parameter (this is the good branch!)
- On the second branch:
  - Contract v and t_1 into a single vertex
  - Apply recursively to the resulting graph without decreasing the parameter (this is the bad branch!)

Polynomially-computable lower bound

- The smallest vertex cut separating $t_1 andt_2, \ldots, t_m$ can be computed in a polynomial time.
- The size of this cut is a lower bound on the size of the minimum multiway cut

The main theorem

Main Result

Assume that contraction of v and t_1 does not increase the lower bound. Then the contraction does not increase the size of the minimum multiway cut as well.

Corollary

In the considered case v and t_1 can be joined without any branching.

The algorithm

`FindCut(G,k)`

1. **If** the terminals are disconnected then **return** 'YES'
2. **If** the lower bound is greater than k then **return** 'NO'
3. **Pick** a terminal t_1 that is not separated from the rest of terminals
4. **Choose** a non-terminal vertex v adjacent to t_1
5. **Contract** v and t_1. Let $G*$ be the resulting graph.
6. **If** separating t_1 from the rest of terminals requires removing the same number of vertices in G and in $G*$ then **return** `FindCut(G*,k)`
7. **If** `FindCut(G,k-1)` returns 'YES' or `FindCut(G*,k)` returns 'YES' then **return** 'YES' **Else return** 'NO'

Runtime analysis

- On the branch where vertex is selected, the parameter decreases.
- On the branch where a vertex is discarded, the lower bound increases (otherwise no branching is performed)
- On each branch the gap between the parameter and the lower bound decreases!
- As a result: the height of the search tree linearly depends on the parameter k.

The Result

Result

The multiway cut problem is FPT.

Reading

Chen, Liu, Lu, "An Improved Parameterized Algorithm for the Minimum Node Multiway Cut Problem", WADS 2007, pp. 495-506

Related reading

Directed Feedback Vertex Set

Chen, Liu, Lu, O'Sullivan, Razgon, "A fixed-parameter algorithm for the directed feedback vertex set problem." Journal of the ACM, 55(5), 2008.

Min 2-CNF Deletion

Razgon,O'Sullivan, "Almost 2-SAT is fixed-parameter tractable", ICALP 2008, pp.551-562.

Outline

The Preprocessing Approach

- The use of bounded search tree methods requires writing essentially new solvers which may be undesirable for some industrial applications.
- For some problems it is possible to transform any algorithm into a fixed-parameter one by writing an easily implementable preprocessing procedure.

The Preprocessing Approach

- The preprocessing procedure transforms the input instance into an equivalent one whose size depends on the parameter k only.
- The resulting instance is called a kernel and the transformation process is called kernelization.
- As a result of kernelization, any solver becomes a fixed-parameter tractable algorithm.

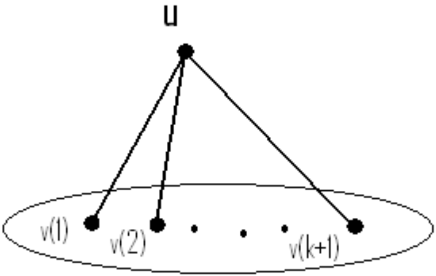
The Preprocessing Approach

- The kernelization is based on reduction rules.
- In their nature, the reduction rules are very similar to achieving arc-consistency for CSP or unit propagation for SAT.
- The difference is that in case of kernelization there is guarantee on the size of the resulting instance (the kernel).

Kernelization of Vertex Cover

Observation

If a graph has a vertex of degree at least $k + 1$, this vertex must be included into the vertex cover. Otherwise, all the neighbors of u are included, i.e. the size of VC exceeds the parameter.



Kernelization of Vertex Cover

Reduction Rule

Whenever there is a vertex of degree at least $k + 1$, remove it from the graph.

Iterative application of the reduction rule results in one of three outcomes.

1. More than k vertices are removed. In this case, return 'NO'.
2. All edges are removed. Then return 'YES'.
3. All the vertices of the resulting graph are of degree at most k.

Kernelization of Vertex Cover

Observation

If max-degree of a graph is at most k and the number of edges is greater than k^2, the graph does not have a vertex cover of size at most k.

Consequence

In the last outcome of the reduction rule, If there are more than k^2 edges, then return 'NO'.

Conclusion

The kernelization process either solves the problem or returns an instance of size $O(k^2)$.

Wrap-up

Where can you get the slides?

Tutorial web-site

`http://www.cs.ucc.ie/~osullb/cp-tutorial-2009/`