

An Introduction to Approximation Algorithms

Adam N. Letchford¹
Lancaster University

Cardiff, January 2010

¹Supported by the EPSRC under grant EP/D072662/1.

Outline

- 1 Motivation and definitions
- 2 Some early *ad hoc* examples
- 3 The search for general principles
- 4 Current status
- 5 Inapproximability

Motivation and Definitions

Many important optimisation problems are hard to solve, either in theory or in practice.

When exact algorithms are incapable of solving instances of interest, one can resort to **heuristics**.

But how can we know whether the heuristic is generating solutions that are 'reasonably close' to optimal?

Sometimes, one can prove a 'performance guarantee' for the heuristic. In such cases, the heuristic is called an **approximation algorithm**.

Motivation and Definitions (cont.)

For approximation algorithms to make any sense, one must make two **key assumptions**:

Assumption 1

The problem is always feasible and one can find a feasible solution in polynomial time.

Assumption 2

The profit (or cost) of the optimal solution is always positive.

Motivation and Definitions (cont.)

Now let $0 \leq \rho < 1$ be a parameter.

Definition (ρ -approximation algorithm, max version)

Consider a maximisation problem satisfying the assumptions. A **ρ -approximation algorithm** is a heuristic that runs in polynomial time and always returns a feasible solution whose profit is at least ρ times the optimal profit.

Motivation and Definitions (cont.)

Now let $0 \leq \rho < 1$ be a parameter.

Definition (ρ -approximation algorithm, max version)

Consider a maximisation problem satisfying the assumptions. A **ρ -approximation algorithm** is a heuristic that runs in polynomial time and always returns a feasible solution whose profit is at least ρ times the optimal profit.

Now suppose instead that $\rho > 1$.

Definition (ρ -approximation algorithm, min version)

Consider a minimisation problem satisfying the assumptions. A **ρ -approximation algorithm** is a heuristic that runs in polynomial time and always returns a feasible solution whose cost is at most ρ times the optimal profit.

Motivation and Definitions (cont.)

Sometimes we can do better...

Definition (PTAS)

A **polynomial-time approximation scheme** (PTAS) is a heuristic that takes ρ as input as well as the instance, runs in polynomial time for fixed ρ , and always finds a ρ -approximate solution.

Definition (FPTAS)

A **fully polynomial-time approximation scheme** (FPTAS) is a PTAS whose running time is bounded by a polynomial of the instance data and $1/\epsilon$, where $\epsilon = 1 - \rho$ for maximisation problems and $\epsilon = \rho - 1$ for minimisation problems.

Some Early *Ad Hoc* Examples

The first approximation algorithms in the literature were mainly based on *ad hoc* arguments.

We will discuss four key early examples:

- Graham's 2-approximation algorithm for machine scheduling.
- Christofides' $3/2$ -approximation algorithm for the TSP.
- Bar-Yehuda & Even's 2-approximation for node cover.
- Ibarra & Kim's FPTAS for the 0-1 knapsack problem.

Early Examples I: Graham (1966)

We wish to assign n jobs to m identical parallel machines. Each job has a fixed processing time.

The goal is to minimise the **makespan** (the time at which the last job is completed).

A natural greedy heuristic is: run through the list of jobs and assign each job to the machine that currently has the least load.

Graham showed that this heuristic is a 2-approximation algorithm.

Early Examples II: Christofides (1976)

The **metric TSP** is the special case of the TSP in which travel costs are symmetric and obey the triangle inequality.

Christofides proposed the following heuristic for the metric TSP:

- Compute a minimum cost spanning tree.
- Let W be the set of nodes with odd degree in the tree.
- Find a minimum cost matching of the nodes in W .
- Combine the edges in the tree and the matching.
- Convert the resulting Eulerian graph into a tour.

He then proved that this is a $3/2$ -approximation algorithm.

Early Examples III: Bar-Yehuda & Even (1981)

Given a graph $G = (V, E)$, the **node cover** problem calls for a node set $W \subset V$ such that every edge in E is incident on at least one node in W .

A trivial heuristic is the following:

- Compute a maximal matching in G .
- Let W contain the end-nodes of the edges in the matching.

Bar-Yehuda & Even (1981) noted that this heuristic is a 2-approximation algorithm.

Early Examples IV: Ibarra & Kim (1975)

The **0-1 knapsack problem** takes the following form:

$$\max \left\{ \sum_{j=1}^n p_j x_j : \sum_{j=1}^n w_j x_j \leq c, x \in \{0, 1\}^n \right\}.$$

It can be solved by dynamic programming in either $\mathcal{O}(nc)$ time or $\mathcal{O}(np^*)$ time, where p^* is the optimal profit.

Ibarra and Kim showed how to modify the latter algorithm, by using 'approximate' profits, to obtain an FPTAS.

The Search for General Principles

Up until the 1980s, all of the known approximation results were *ad hoc* (tailored to one specific problem).

It would of course be nice to have some **general principles** for constructing approximation algorithms.

No fully general method has been found, but some useful frameworks have emerged.

Examples include *randomised rounding*, the *primal-dual schema*, *approximate dynamic programming* and *divide-and-conquer*.

General Principles I: Randomised Rounding

This was developed by Raghavan and colleagues in the mid-1980s. The basic idea is:

- Formulate the problem as an Integer Linear Program.
- Solve the LP relaxation of the ILP.
- Randomly round fractional variables to integers (using the LP values as a guide).
- Repair the solution if necessary to achieve feasibility.

This method has worked well for multi-commodity flow problems, network design problems, and covering problems.

General Principles I: Randomised Rounding (cont.)

Remark

One does not have to use linear relaxations. Non-linear relaxations can work well too!

For example:

- Goemans & Williamson (1995) used **semidefinite programming** to get 0.878-approximation algorithms for max-cut and max-2-sat.
- Skutella (2001) used **convex quadratic programming** to derive a $3/2$ -approximation algorithm for parallel machine scheduling.

General Principles II: the Primal-Dual Schema

This was developed by Vazirani and colleagues in the mid-1990s.

It is based on classical primal-dual algorithms for LP, which iteratively construct primal and dual solutions that satisfy the *complementary slackness* conditions.

To obtain an approximation algorithm, one has to ensure that the primal solution is integral, and that the complementary slackness conditions are 'approximately' satisfied.

This schema has yielded good approximation algorithms for various facility location and network design problems.

General Principles III: Approximate Dynamic Programming

Many hard combinatorial optimisation problems (e.g., knapsack, lot-sizing, scheduling), can be solved in pseudo-polynomial time by dynamic programming.

When this is the case, it is frequently possible to convert the dynamic programme into an FPTAS.

There are two main ways to do it: *simplify the input data* (as in Ibarra & Kim, 1975) or *trim the state space* (as in Sahni, 1976).

These ideas have been made rigorous by Woeginger (2000).

General Principles IV: Divide-and-Conquer

For some problems, it has been possible to construct a PTAS via 'divide-and-conquer':

- Break the problem into lots of small 'pieces'
- Solve optimally a subproblem for each 'piece'
- Optimally 'glue together' the solutions to the subproblems.

An important example is the PTAS for the Euclidean TSP, due to Arora (1998). It finds a $(1 + \epsilon)$ -approximate solution in $\mathcal{O}(n^{1/\epsilon})$ time.

(PTASs, unlike FPTASs, are often of only theoretical interest!)

Current Status

By the end of the 1970s, hundreds of problems had been categorised as either polynomially-solvable or \mathcal{NP} -hard.

By the end of the 1990s, hundreds of \mathcal{NP} -hard problems had been placed into one of four categories:

- Having an FPTAS
- Having a PTAS
- Approximable to some constant
- Probably not approximable at all.

Current Status (cont.)

Which problems have an FPTAS?

Current Status (cont.)

Which problems have an FPTAS?

Most (but not all) problems that can be solved in pseudo-polynomial time (Woeginger)

Current Status (cont.)

Which problems have an FPTAS?

Most (but not all) problems that can be solved in pseudo-polynomial time (Woeginger)

Which problems have a PTAS?

Current Status (cont.)

Which problems have an FPTAS?

Most (but not all) problems that can be solved in pseudo-polynomial time (Woeginger)

Which problems have a PTAS?

Various problems involving points in Euclidean space (Arora)
Various problems involving 'dense' graphs or matrices (Arora)
Some other miscellaneous problems

Current Status (cont.)

Best known constants for some problems:

- Metric Traveling Salesman: $3/2$
- Node Cover: 2
- Steiner Tree: 1.549
- Simple Plant Location: 1.4998
- Survivable Network Design: 2
- Max-Cut: 0.878
- Max-Di-Cut: 0.874
- Max-2-Sat: 0.940
- Max-Sat: 0.8331
- Linear Ordering: $1/2$

Current Status (cont.)

Some problems that have **resisted approximation**:

- Linear arrangement: $\sqrt{\log n} \log \log n$
- Set covering: $\log n$
- Chromatic number: $n/(\log n)^3$
- Shortest/closest vector: $2^{n/\log n}$
- Max-Clique: $(\log n)^2/n$

Inapproximability

It is sometimes possible to prove that a problem **cannot be approximated** to within a certain factor (if $\mathcal{P} \neq \mathcal{NP}$).

Results of this kind are called **inapproximability** results.

There have been three major developments in inapproximability:

- Creation of MAX-SNP by Papadimitriou & Yannakakis (1988)
- The PCP-theorem by Arora *et al.* (1992) and its use by Håstad (1997).
- The unique games conjecture of Khot (2002).

Inapproximability (cont.)

Papadimitriou & Yannakakis (1988) defined a new complexity class called MAX-SNP and a new kind of problem reduction called *L-reduction*.

They proved that a variety of problems (TSP, max-clique, set covering...) were MAX-SNP-complete.

They then proved that a MAX-SNP-complete problem cannot have a PTAS, if $\mathcal{P} \neq \mathcal{NP}$.

Inapproximability (cont.)

Arora *et al.* (1992) gave a new characterisation of \mathcal{NP} based on so-called ‘probabilistically checkable proofs’.

They then used this to show that some problems cannot be approximated to within *any* constant factor (if $\mathcal{P} \neq \mathcal{NP}$).

This approach was honed by Håstad (1997) to get quite tight bounds.

E.g., max-cut cannot be approximated better than $16/17$ (≈ 0.941).

Inapproximability (cont.)

Khot (2002) made two (plausible) conjectures and showed that their truth would imply tight bounds on the approximability of a range of problems.

E.g., it would be hard to approximate max-cut and node cover better than 0.878 and 2, respectively.

The first conjecture, the *majority-is-stablest* conjecture, was proved by Mossel *et al.* in 2005.

The other conjecture, the *unique games* conjecture, is a major open question in theoretical computer science.

Conclusion

Approximation algorithms are a special kind of heuristic that have an *a priori* guarantee on their performance.

Although some general frameworks exist for constructing them, one must still study and exploit the structure of the problem at hand.

Some \mathcal{NP} -hard problems are ‘easier’ than others.

Approximation and inapproximability are very hot topics in theoretical computer science — see recent FOCS, SODA and STOC proceedings.