Approximations through relaxations

GERHARD J. WOEGINGER *

Abstract

We discuss polynomial time approximation results for two combinatorial optimization problems (one problem from graph theory, one problem from scheduling theory). The results are based on the technique of rounding the optimal solution of an underlying linear programming relaxation. We analyze these relaxations, their integrality gaps, and the resulting approximation algorithms, and we derive matching worst case instances.

Keywords: Approximation algorithm; worst case analysis; performance guarantee; linear programming relaxation; integrality gap

1 Introduction

Most real-world optimization problems are NP-hard. And most NP-hard problems are difficult to solve to optimality. We conclude: Most real-world optimization problems are difficult to solve to optimality. A standard way of working around this rather pessimistic conclusion is to relax *optimality*, and to satisfy oneself instead with *near-optimal* or *approximate* solutions. This leads us into the area of approximation algorithms for combinatorial optimization problems.

A combinatorial optimization problem consists of a set $\mathcal{I}$ of instances, and a family $\mathcal{F}(I)$ of feasible solutions for every instance $I \in \mathcal{I}$. Every feasible solution $F \in \mathcal{F}(I)$ comes with a non-negative cost $c(F)$. In this paper, we will only consider minimization problems, where the objective is to determine a feasible solution of minimum possible cost. An *approximation algorithm* is an algorithm that for every instance $I \in \mathcal{I}$ returns a near-optimal solution. If it manages to do this in polynomial time, then it is called a *polynomial time* approximation algorithm. An approximation algorithm for a minimization problem is called a *ρ-approximation algorithm*, if it always returns a near-optimal solution with cost at most a factor ρ above the optimal cost. Such a value $\rho \geq 1$ is called a *worst case performance guarantee* of the algorithm.

Approximations through relaxations. One standard approach for designing polynomial time approximation algorithms for a (difficult, NP-hard) optimization problem $\mathcal{P}$ is the following:

(S1) Relax some of the constraints of the hard problem $\mathcal{P}$ to get an easier problem $\mathcal{P}'$ (the so-called *relaxation*).

(S2) Compute in polynomial time an optimal solution S' for this easier relaxed problem $\mathcal{P}'$.

(S3) Translate in polynomial time the solution S' into an approximate solution S for the original problem $\mathcal{P}$.

(S4) Analyze the quality of solution S for $\mathcal{P}$ by comparing its cost to the cost of solution S' for $\mathcal{P}'$.

*gwoegi@win.tue.nl. Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven.

Let C^{Opt} denote the optimal cost of the orginal problem instance, let C^{Rlx} denote the optimal cost of the relaxed instance, and let C^{App} denote the cost of the translated approximate solution. To show that the sketched approach has a performance guarantee of ρ, one usually establishes the following chain of inequalities:

$$C^{Rlx} \leq C^{Opt} \leq C^{App} \leq \rho \cdot C^{Rlx} \leq \rho \cdot C^{Opt}. \tag{1}$$

The first and the last inequality in this chain are trivial, since problem $\mathcal{P}'$ results from problem $\mathcal{P}$ by relaxing constraints. The second inequality is also trivial, since the optimal solution is at least as good as some approximate solution. The third inequality contains the crucial step in the chain, and all the analysis work goes into proving that step. This third inequality relates the relaxed solution to the approximate solution; both solutions are polynomial time computable, and hence their combinatorics will be nice and well-behaved. Thus, the chain yields the desired relation $C^{App} \leq \rho \cdot C^{Opt}$ by analyzing nice, polynomial time computable objects. The analysis avoids touching the orginal NP-hard problem whose combinatorics is messy and complicated and hard to grasp.

Worst case gaps and integrality gaps. Of course, we would like to make the value of the parameter ρ as small as possible: The closer ρ is to 1, the better is the performance of the approximation algorithm. How can we argue that our worst case analysis is complete? How can we argue that we have reached the smallest possible value for ρ? That's usually done by exhibiting a so-called *worst case instance*, that is, an instance I that demonstrates a *worst case gap* of ρ for the approximation algorithm:

$$C_I^{App} = \rho \cdot C_I^{Opt} \qquad \text{and} \qquad C_I^{Opt} = C_I^{Rlx}. \tag{2}$$

Here the left hand equation establishes the gap, and together with the chain (1) it yields the right hand equation. The worst case instance (2) illustrates that our analysis of the *combined* approach (S1)–(S3) is tight. Is this the end of the story? Not necessarily. We could possibly start with the same relaxation step (S1), then solve the relaxation with the same step (S2), and then come up with a completely new (and better) translation step. How can we argue that this is not possible? How can we argue that the value ρ is already the best performance guarantee that we possibly can get out of the considered relaxation? That's usually done by exhibiting an instance J that demonstrates an *integrality gap* of ρ between the original problem and the relaxation:

$$C_J^{Opt} = \rho \cdot C_J^{Rlx} \qquad \text{and} \qquad C_J^{Opt} = C_J^{App}. \tag{3}$$

The equation on the left hand side establishes the gap, and with (1) it yields the equation on the right hand side. In particular, we have $C_J^{App} = \rho \cdot C_J^{Rlx}$. For instance J the third inequality in the chain (1) is tight, and there is no way of proving a better performance guarantee for an approximation algorithm built around the considered relaxation. We stress that such a better approximation algorithm around the relaxation might well exist, but we will never be able to *prove* that its performance guarantee is better than ρ within the framework described above.

For $\rho > 1$, the conditions in (2) and in (3) can not be satisfied by the same instance, as they would be contradictory. Hence, a complete analysis of an approximation algorithm within this framework always must provide *two* separate bad instances, one for the worst case gap and one for the integrality gap.

Overview of this paper. We will illustrate the approach (S1)–(S4) with two examples. The first example (in Section 2) comes from graph theory, and the second example (in Section 3) comes from scheduling theory. For both examples we provide an integer programming formulation, a relaxation, an approximation algorithm, a worst case analysis, and two gap instances. For more information on this field, we refer the reader to the excellent book [6] by Vazirani.

2 The vertex cover problem

As our first example we will discuss the *vertex cover* problem (VC, for short): An input to this problem consists of an undirected graph $G = (V, E)$. A subset $S \subseteq V$ that touches every edge in E is called a *vertex cover* of G, and the goal is to find a vertex cover of minimum cardinality. Problem VC is well-known to be NP-hard (Garey & Johnson [2]).

Exact formulation and relaxation. Consider the following integer programming formulation (4) of VC. Let $v_1, \ldots, v_n$ be an enumeration of the vertices in V. For every vertex v_i, the binary variable x_i decides whether v_i is in the vertex cover (in which case $x_i = 1$) or whether v_i is not in the vertex cover ($x_i = 0$).

$$\begin{array}{lll} \min & \sum_{i=1}^{n} x_i & \\ \text{s.t.} & x_i + x_j \geq 1 & \text{for every edge } [v_i, v_j] \in E \\ & x_i \in \{0, 1\} & \text{for } i = 1, \ldots, n \end{array} \tag{4}$$

The constraints state that every edge must be touched by the vertices v_i with $x_i = 1$, that is, by the vertices in the cover. Since VC is an NP-hard problem, also the equivalent integer programming formulation (4) will be NP-hard to solve. Therefore, we relax this formulation to get something simpler: We replace the integrality constraints "$x_i \in \{0, 1\}$" by continuous constraints "$0 \leq x_i \leq 1$":

$$\begin{array}{lll} \min & \sum_{i=1}^{n} x_i & \\ \text{s.t.} & x_i + x_j \geq 1 & \text{for every edge } [v_i, v_j] \in E \\ & 0 \leq x_i \leq 1 & \text{for } i = 1, \ldots, n \end{array} \tag{5}$$

Since all variables are now continuous, this relaxation is a standard linear program and can be solved to optimality in polynomial time. We denote the optimal objective value of (4) by C^{Opt}, and the optimal objective value of the LP-relaxation (5) by C^{LP}. Furthermore, we denote by x_i^{LP} the value of variable x_i in the optimal LP-solution.

The approximation algorithm. Now let us translate the LP-solution into a 'nearby' feasible IP-solution. The trouble with the LP-solution is that the variables x_i^{LP} need not be integral, whereas we would like them to be binary. A simple way of resolving this problem is to do threshold rounding: If $x_i^{LP} < 1/2$, then we define a corresponding *rounded* variable $\tilde{x}_i = 0$. And if $x_i^{LP} \geq 1/2$, then we define a corresponding rounded variable $\tilde{x}_i = 1$. The corresponding *rounded* objective value is denoted by $C^{App} = \sum_{i=1}^{n} \tilde{x}_i$.

Let us verify that the rounded values $\tilde{x}_i$ constitute a feasible solution of (4): If they violate some constraint "$\tilde{x}_i + \tilde{x}_j \geq 1$", then $\tilde{x}_i = \tilde{x}_j = 0$ must hold, and thus $x_i^{LP} < 1/2$ and $x_j^{LP} < 1/2$. But this would yield $x_i^{LP} + x_j^{LP} < 1$, and contradict the feasibility of the LP-solution for (5). Hence, the rounded solution indeed is feasible for (4). What about its quality? We observe that the threshold rounding yields

$$\tilde{x}_i \leq 2\, x_i^{LP} \qquad \text{for } i = 1, \ldots, n. \tag{6}$$

By adding up the inequalities (6) for $i = 1, \ldots, n$ we derive that

$$C^{App} = \sum_{i=1}^{n} \tilde{x}_i \leq 2 \sum_{i=1}^{n} x_i = 2\, C^{LP} \leq 2\, C^{Opt}. \tag{7}$$

Hence, the described approximation algorithm has a worst case performance guarantee of at most 2.

Analysis of the two gaps. Is our worst case bound of 2 on the worst case performance guarantee of this algorithm best possible? Yes, it is: Consider the complete bipartite graph where $V = L \cup R$ with $|L| = |R| = n/2$ and $E = \{[u, v] : u \in L, v \in R\}$. Then an optimal vertex cover consists of all vertices in L, and has $C^{Opt} = n/2$. One optimal LP-solution is given by $x_i^{LP} \equiv 1/2$; this leads to $\tilde{x}_i \equiv 1$ and $C^{App} = n$.

Gap 2.1 *There exist instances for VC for which the gap between the optimal objective value and the objective value produced by the rounding algorithm equals* 2.

And what about the integrality gap of the LP-relaxation? Consider the complete graph K_n on n vertices (the graph that contains all possible edges). Then $C^{Opt} = n - 1$, since by omitting two vertices, we would not touch the edge between these two vertices. Morevoer, the LP-relaxation has an optimal solution with $x_i^{LP} \equiv 1/2$ and $C^{LP} = n/2$.

Gap 2.2 *The integrality gap of the linear programming relaxation for problem VC is* 2.

Arora, Bollobás & Lovász [1] show that certain large families of linear programming relaxations for vertex cover all have integrality gaps of at least 2. It is an outstanding open problem to break through this barrier of 2. Håstad [3] has proved that unless P=NP, there cannot be a polynomial time approximation algorithm for vertex cover with performance guarantee strictly less than 7/6.

3 Scheduling with job rejections

In this section, we consider an environment with n jobs $J_1, \ldots, J_n$ and with m *unrelated* parallel machines $M_1, \ldots, M_m$. Job J_j has a processing time p_{ij} on machine M_i, and moreover job J_j has a positive *rejection penalty* f_j. All jobs are available at time 0. Preemption of the jobs is allowed (that is, a job may be arbitrarily interrupted and resumed later on an arbitrary machine). A job may be processed on at most one machine at a time, and every machine may process at most one job at a time. For each job J_j, it must be decided whether to accept or to reject it. The accepted jobs are to be scheduled on the m machines. For the accepted jobs, we pay the makespan of this schedule (that is, the maximum job completion time). For the rejected jobs, we pay their rejection penalties. In other words, the objective value is the preemptive makespan of the accepted jobs plus the total penalty of the rejected jobs. This scheduling problem is denoted by SCHED. Hoogeveen, Skutella & Woeginger [4] have proved that it is NP-hard in the strong sense.

Exact formulation and relaxation. Again, we will start by stating an integer programming formulation. For job J_j, the binary variable y_j decides whether J_j is rejected ($y_j = 0$) or accepted ($y_j = 1$). The continuous variables x_{ij} describe which percentage of job J_j should be processed on machine M_i. The continuous variable C denotes the optimal preemptive makespan for the accepted jobs.

$$
\begin{array}{lll}
\min & C + \sum_{j=1}^{n} (1 - y_j) f_j & \\
\text{s.t.} & \sum_{j=1}^{n} x_{ij} p_{ij} \leq C & \text{for } i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} p_{ij} \leq C & \text{for } j = 1, \ldots, n \\
& \sum_{i=1}^{m} x_{ij} = y_j & \text{for } j = 1, \ldots, n \\
& x_{ij} \geq 0 & \text{for } i = 1, \ldots, m \text{ and } j = 1, \ldots, n \\
& y_j \in \{0, 1\} & \text{for } j = 1, \ldots, n
\end{array}
\tag{8}
$$

The first family of constraints states that for every machine the total assigned processing time is at most C. The second family of constraints states that the total processing time of any accepted

job cannot exceed C. The third family of constraints connects the binary decision variables y_j to the continuous variables x_{ij}: If a job is accepted ($y_j = 1$), then the percentages x_{ij} should add up to $1 = y_j$. If a job is rejected ($y_j = 0$), then the percentages x_{ij} should add up to $0 = y_j$.

As soon as we have fixed all the values x_{ij}, the remaining makespan minimization problem is essentially makespan minimization in a preemptive open shop. It is well-known [5] that for a preemptive open shop, the smallest value C fulfilling the first and second family of constraints in (8) yields the optimal preemptive makespan. To summarize, the integer program (8) is a complete and correct description of the problem SCHED.

We define the LP-relaxation of the integer programming formulation (8) by replacing the integrality constraints "$y_j \in \{0,1\}$" by continuous constraints "$0 \leq y_j \leq 1$". This LP-relaxation can be solved in polynomial time, and we denote an optimal solution by x_{ij}^{LP}, y_j^{LP}, and C^{LP}.

The approximation algorithm. Now we want to translate the LP-solution into a reasonable feasible solution for the (8). We mainly have to take care of the decision variables y_j; however, this time we also must pay attention to the continuous variables x_{ij}, since their values depend on the values y_j via the third family of constraints in (8).

We *randomly* choose a threshold α from the uniform distribution over the interval $[1/e, 1]$; here as usual $e \approx 2.71828$ denotes the base of the natural logarithm. If $y_j^{LP} \leq \alpha$, then we define a rounded decision variable $\tilde{y}_j := 0$, and otherwise we define $\tilde{y}_j := 1$. Jobs J_j with $\tilde{y}_j = 0$ are rejected in the rounded solution, and we set all their variables $\tilde{x}_{ij} = 0$. Jobs J_j with $\tilde{y}_j = 1$ are accepted in the rounded solution; we set all their variables $\tilde{x}_{ij} := x_{ij}^{LP}/y_j^{LP}$. Finally, we define the rounded makespan by

$$\tilde{C} \;:=\; \max\{\max_{1\leq i\leq m}\sum_{j=1}^{n}\tilde{x}_{ij}p_{ij},\;\; \max_{1\leq j\leq n}\sum_{i=1}^{m}\tilde{x}_{ij}p_{ij}\}. \tag{9}$$

It can be verified that the rounded solution $\tilde{x}_{ij}$, $\tilde{y}_j$, and $\tilde{C}$ constitutes a feasible solution of (8): All variables $\tilde{y}_j$ are binary. For j with $\tilde{y}_j = 0$, the variables $\tilde{x}_{ij}$ add up to 0. For j with $\tilde{y}_j = 1$, the variables $\tilde{x}_{ij}$ add up to $\sum_i x_{ij}^{LP}/y_j^{LP} = 1$. Finally, in (9) the value of $\tilde{C}$ is fixed in order to fulfill the first and the second family of constraints.

Now let us analyze the quality of this rounded solution. For any fixed value of α, the rounded variable $\tilde{x}_{ij}$ is at most a factor of $1/\alpha$ above x_{ij}^{LP}. Hence, by linearity also $\tilde{C}$ is at most a factor of $1/\alpha$ above C^{LP}. Then the expected multiplicative increase in the makespan is at most a factor of

$$\frac{e}{e-1}\int_{1/e}^{1} 1/\alpha \; d\alpha \;=\; \frac{e}{e-1}\;.$$

In the LP-solution, the contribution of job J_j to the total rejection penalty is $(1 - y_j^{LP})f_j$. The expected contribution of J_j to the rejection penalty in the rounded solution is

$$\begin{aligned} f_j \cdot \mathrm{Prob}[y_j^{LP} \leq \alpha] \;&=\; f_j \int_{\max\{1/e,\, y_j^{LP}\}}^{1} \frac{e}{e-1} d\alpha \\ &\leq\; f_j \int_{y_j^{LP}}^{1} \frac{e}{e-1} d\alpha \\ &=\; \frac{e}{e-1}\cdot(1-y_j^{LP})f_j. \end{aligned}$$

All in all, the expected objective value for the rounded solution is at most a factor of $e/(e-1) \approx 1.58$ above the optimal objective value of the LP-relaxation. Hence, our procedure yields a *randomized* polynomial time approximation algorithm with a worst case performance guarantee of $e/(e-1)$.

How can we turn this randomized algorithm into a deterministic algorithm? Well, the only critical values for the threshold parameter α are the values y_j^{LP} with $j = 1, \ldots, n$. All other values of α will yield the same solution as for one of these criticial values. Hence, it is straightforward to derandomize the algorithm in polynomial time: We compute the n rounded solutions that correspond to these n critical values, and we select the solution with smallest objective value.

Analysis of the two gaps. Our next goal is to give a worst case instance for the above approximation algorithm for SCHED. The instance is based on an integer parameter q. There are $m = (q+1)^q - q^q$ machines M_j that are indexed by $j = q^q + 1, \ldots, (q+1)^q$. For every machine M_j, there are two corresponding jobs J_j and J'_j that have infinite processing requirements on all other machines M_i with $i \neq j$; this implies that these two jobs either have to be rejected or have to be processed on M_j. The processing time of job J_j on M_j is $j - q^q$, and its rejection penalty is $(j - q^q)/j$. The processing time of job J'_j on M_j is q^q, and its rejection penalty is q^q/j. Note that the overall processing time of J_j and J'_j is j, and that their overall penalty is 1.

One possible feasible solution accepts all the jobs J'_j and rejects all the jobs J_j. The resulting makespan is $C = q^q$, and the resulting objective value equals

$$q^q + \sum_{j=q^q+1}^{(q+1)^q} (j - q^q)\frac{1}{j} \;=\; (q+1)^q - q^q \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j} \;. \tag{10}$$

It can be verified that this in fact is the optimal objective value. Next, assume that the approximation algorithm starts from the following feasible solution for the LP-relaxation: For $j = q^q + 1, \ldots, (q+1)^q$ the two jobs J_j and J'_j both get an acceptance value $y_j^{LP} = q^q/j$. Then on every machine M_j, the overall accepted processing time is $(j - q^q)y_j^{LP} + q^q\, y_j^{LP} = q^q$. The penalty of job J_j plus the penalty of job J'_j equals $1 - y_j^{LP} = (j - q^q)/j$. Hence, the objective value of this LP-solution equals the value in (10).

Consider the rounding step for some fixed threshold α. Since the values $y_j^{LP} = q^q/j$ are decreasing in j, there exists some index k such that for $j \leq k$ the values $y_j^{LP} = q^q/j$ all are rounded up to 1 (and the corresponding jobs are accepted), whereas for $j \geq k+1$ the values $y_j^{LP} = q^q/j$ all are rounded down to 0 (and the corresponding jobs are rejected). Then the makespan becomes k (the load on machine M_k), the total rejection penalty is $(q+1)^q - k$, and the objective value is $(q+1)^q$. Thus, the objective value in the rounded solution always equals $(q+1)^q$, and does not depend on α or k.

The ratio between the optimal objective value in (10) and the approximate objective value of $(q+1)^q$ equals

$$1 - \left(\frac{q}{q+1}\right)^q \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j} \;. \tag{11}$$

We estimate the sum in (11) by

$$\int_{q^q+1}^{(q+1)^q+1} \frac{1}{z}\, dz \;\leq\; \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j} \;\leq\; \int_{q^q}^{(q+1)^q} \frac{1}{z}\, dz \;.$$

Since the integrals on the left and on the right hand side both converge to 1 when q goes to infinity, the same holds true for the sum in between. Hence, the ratio in (11) behaves roughly like $((q+1)^q - q^q)/(q+1)^q$. For large q, this ratio tends to $(e-1)/e$.

Gap 3.1 *There exist instances of SCHED for which the gap between the optimal makespan and the makespan produced by the rounding algorithm comes arbitrarily close to* $e/(e-1)$.

Our final goal in this section is to get the matching lower bound of $e/(e-1)$ for the integrality gap of the LP-relaxation for SCHED. We use a slight modification of the instance constructed above. Again, we use an integer parameter q, and again there are $m = (q+1)^q - q^q$ machines M_j that are indexed by $j = q^q + 1, \ldots, (q+1)^q$. For every machine M_j, there is one corresponding job J_j. The processing requirements of J_j are $p_{jj} = j$, and $p_{ij} = \infty$ for $i \neq j$. The rejection penalty is uniformly $f_j \equiv 1$.

Consider a 'reasonable' feasible schedule with makespan T: Then the jobs on machines M_j with $j \leq T$ will be accepted, and the jobs on machines M_j with $j > T$ will be rejected. This yields a total rejection penalty of $(q+1)^q - T$, and an optimal objective value of $(q+1)^q$. Next, consider

the following feasible solution for the LP-relaxation: We set $y_j^{LP} = q^q/j$ for $j = q^q+1, \ldots, (q+1)^q$, and we set $C^{LP} = q^q$. The objective value of this solution is equal to

$$q^q + \sum_{j=q^q+1}^{(q+1)^q} (1 - y_j^{LP}) \quad = \quad (q+1)^q - q^q \sum_{j=q^q+1}^{(q+1)^q} \frac{1}{j} \ .$$

Since this LP-value is equal to the value in (10), and since the optimal value is equal to $(q+1)^q$, the ratio of these two values equals the ratio in (11). The arguments around (11) show that as q becomes large, this ratio tends to $(e-1)/e$.

Gap 3.2 *For problem SCHED, the integrality gap of the LP-relaxation is* $e/(e-1)$.

It would be nice to get a polynomial time approximation algorithm for SCHED with a worst case performance guarantee better than $e/(e-1)$.

References

[1] S. Arora, B. Bollobás, and L. Lovász (2002). Proving integrality gaps without knowing the linear program. *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'2002)*, 313–322.

[2] M.R. Garey and D.S. Johnson (1979). *Computers and Intractability.* W.H. Freeman and Co., New York.

[3] J. Håstad (1999). Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica 182*, 105–142.

[4] H. Hoogeveen, M. Skutella, and G.J. Woeginger (2003). Preemptive scheduling with rejection. *Mathematical Programming 94*, 361–374.

[5] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (1993). Sequencing and scheduling: Algorithms and complexity. In: S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.) *Logistics of Production and Inventory*, Handbooks in Operations Research and Management Science 4, North-Holland, Amsterdam, 445 522.

[6] V.V. Vazirani (2001). *Approximation Algorithms.* Springer-Verlag, Berlin, Heidelberg.