2 Approximation with absolute guarantees

Definition 2.1 Let Π be an optimization problem and A be an approximation algorithm for Π.

(a) The *absolute error* of A with respect to instance I is

$$\kappa_A(I) = |A(I) - \mathrm{OPT}(I)| \ .$$

(b) The *absolute worst-case error* of A is the function

$$\kappa_A^{wc}(n) = \max\{\kappa_A(I) : \ I \in \mathcal{D}, \ |I| \leq n\} \ .$$

(c) Let $\kappa_A : \mathbb{N}_0 \to \mathbb{N}_0$ be a function. A *guarantees* an absolute error of $\kappa_A(n)$ if for all n, $\kappa_A^{wc}(n) \leq \kappa_A(n)$.

(d) Let $\kappa'_A : \mathbb{N}_0 \to \mathbb{N}_0$ be a function. A has an *absolute deviation* of $\kappa'_A(n)$ if for infinitely many n, $\kappa'_A(n) \leq \kappa_A^{wc}(n)$.

It would be nice if the absolute error could be kept very small. As we will see, this cannot always be achieved. In fact, we will show that there are problems with the property that if they had an approximation algorithm with constant absolute worst-case error, then $\mathbf{P} = \mathbf{NP}$.

2.1 Graph coloring

Let $G = (V, E)$ be an undirected graph. For $u \in V$ let $\Gamma_G(u) = \{v : \ \{u, v\} \in E\}$ be the set of *neighbors* of u and $\deg_G(u) = |\Gamma_G(u)|$ be the *degree* of u. The degree of G is $\Delta(G) = \max\{\deg_G(u) : \ u \in V\}$. G is called *r-regular* if $\deg_G(u) = r$ for all nodes $u \in V$. If there is no ambiguity about which graph is meant, we will omit the index G. Given a function $f : N \to M$ and a subset $U \subseteq N$, we define $f(U) = \{y \in M : \text{there exists an } x \in U \text{ with } f(x) = y\}$.

Definition 2.2 Let $G = (V, E)$ be an undirected graph.

(a) A mapping $c_V : V \to \mathbb{N}$ is called a *node coloring* of G if for all $\{u, v\} \in E$, $c_V(u) \neq c_V(v)$.

(b) A mapping $c_E : E \to \mathbb{N}$ is called an *edge coloring* of G if for all edges $e, e' \in E$ with a common node, $c_E(e) \neq c_E(e')$.

$c_V(u)$ and $c_E(e)$ are called *colors*. $|c_V(V)|$ and $|c_E(E)|$ denote the amount of colors used.

These definitions allow to formulate the following optimization problems (we will be that formal for the last time).

Definition 2.3 The *node or edge coloring problem* is characterized by

- $\mathcal{D} = \{\langle G \rangle : \ G = (V, E) \text{ is an undirected graph with at least one edge}\}$
- $S(\langle G \rangle) = \{c_V : \ c_V \text{ is a node coloring of } G\}$ resp. $S(\langle G \rangle) = \{c_E : \ c_E \text{ is an edge coloring of } G\}$
- $f(c_V) = |c_V(V)|$ resp. $f(c_E) = |c_E(E)|$
- min

That is, the goal is to find for a given graph $G = (V, E)$ a node coloring resp. edge coloring that uses as few colors as possible.

In the case of node colorings, the smallest possible number of colors is called the *chromatic number* of G and denoted by $\chi(G)$. In the case of edge colorings, the smallest possible number of colors is called the *chromatic index* of G and denoted by $\chi'(G)$.

The decision problem "Is the graph G node colorable with k colors?" is **NP**-complete for all $k \geq 3$ [GJ79, p. 191]. The same holds for the question "Is the graph G edge colorable with k colors?" [Hol81].

Remark 2.4

(a) There are graphs G that need at least $\Delta(G) + 1$ colors for a node coloring.

(b) There are graphs G that need at least $\Delta(G) + 1$ colors for an edge coloring.

The proof of the remark is an easy exercise. In the following, we will present algorithms that require for both node and edge colorings at most $\Delta(G) + 1$ colors.

2.2 Node colorings

Arbitrary graphs

Consider a graph $G = (V, E)$ with $V = \{u_1, \ldots, u_n\}$. The following algorithm computes a feasible node coloring for G.

Algorithm GreedyCol

```
for i = 1 to n do c_v(u_i) = ∞
for i = 1 to n do
    c_V(u_i) = min(IN \ {c_V(Γ(u_i))})
ouput c_V
```

The algorithm works as follows: we use the value ∞ to identify a node as non-colored. The nodes will be colored in the order $u_1, \ldots, u_n$. Node u_i gets the smallest color that is not already used by one of its neighbors. GreedyCol is a typical representative of the class of greedy algorithms.

Theorem 2.5 *Let $G = (V, E)$ be a graph. Algorithm* GreedyCol *computes in time $O(|V| + |E|)$ a node coloring with at most $\Delta(G) + 1$ colors, that is,* $\textsc{GreedyCol}(G) \leq \Delta(G) + 1$.

Proof. It is easy to implement the algorithm with the claimed runtime. When the algorithm has reached u_i, not all of the colors can be given to the $\deg(u_i)$ neighbors of u_i, since there are more colors available than u_i has neighbors. Hence, there must be a smallest available color in the set $\{1, \ldots, \deg(u_i) + 1\}$ that can be assigned to u_i. □

Since any graph G with at least one edge has to use at least two different colors, i.e. $\mathrm{OPT}(G) \geq 2$, we immediately obtain the following result.

Theorem 2.6 *Algorithm* GreedyCol *guarantees an absolute error of*

$$\kappa_{\textsc{GreedyCol}}(G) = \textsc{GreedyCol}(G) - \mathrm{OPT}(G) \leq \Delta(G) + 1 - 2 = \Delta(G) - 1 \ .$$

This statement demonstrates that the error may not necessarily be a function of the length of the input. Here, for example, we use as a parameter the degree of G that cannot be directly obtained from the length of the input.

How good is now the error of the analysis? What about the absolute deviation? We only showed so far that GreedyCol uses at most $\Delta(G) + 1$ colors. It could be that our analysis is too weak, or that the algorithm always uses the minimum number of colors (and therefore has a deviation of 0). It can be shown that this is not the case by providing an infinite set of graphs $\mathcal{G} = \{G_1, G_2, \ldots\}$ with the property that for every graph G_i there is a computation of GreedyCol that uses $\Delta(G_i) + 1$ colors although G_i can be colored with only a few colors.

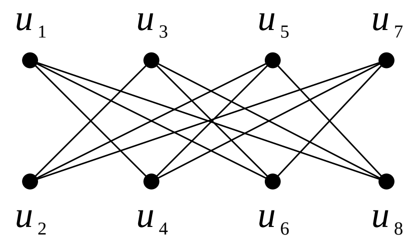

Figure 1: An input for which GreedyCol computes a bad output.

Consider as an example the graph given in Figure 1. The numbering of the nodes causes the greedy algorithm to use $\Delta(G)+1$ colors although it is node-2-colorable. This graph can be easily extended to a set of graphs $\mathcal{G}$ with the above properties. In other words: GreedyCol has an absolute deviation of $\Delta(G) - 1$. The statement in Theorem 2.6 is therefore sharp.

What causes GreedyCol to fail? One look at the set of graphs $\mathcal{G}$ shows that the naming of the nodes was bad for the algorithm. Had the nodes in the upper row been given numbers before the nodes in the lower row, GreedyCol would have found an optimal solution. It is in fact possible for *every* graph to name the nodes so that GreedyCol provides an optimal solution.

The degree of the graph G can depend on the number of nodes. It can be even $\Theta(|V|)$. In this case, the absolute deviation of GreedyCol can be very large. In Section 3.2 we will present a greedy algorithm that only needs $O(|V|/\log|V|)$ colors. Much better methods are not known yet, only a very complex but only slightly better algorithm [Hal93]. In the following subsection we will see that much better results can be obtained for planar graphs.

Planar graphs

A graph $G = (V, E)$ is called *planar* if it can be drawn on the plane without crossings between edges, or equivalently, if one can embed the nodes on the surface of a sphere so that the edges connecting the nodes do not intersect. A region of the surface that is bounded by edges and does not contain an edge or node is called a *facette*. Figure 2 shows a planar and two non-planar graphs. The planar graph has 7 facettes (the area around the graph also counts as a facette). Note that one really has to prove that the other graphs in Figure 2 (these are the so-called K_5, a complete graph on 5 nodes, and the $K_{3,3}$, a complete bipartite graph with 3 nodes on either side) are not planar.

In the following, we consider the problem to color the nodes of a planar graph, i.e. we modify the set $\mathcal{D}$ in Definition 2.3 so that the graphs have to be planar. The following facts list important results about the colorability of planar graphs.

Fact 2.7

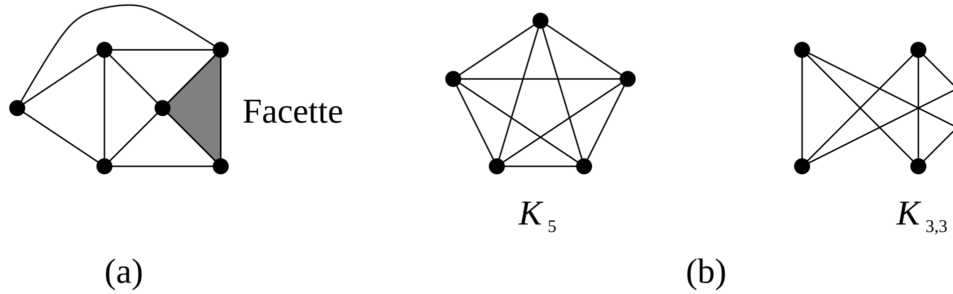

Figure 2: (a) A planar and (b) two non-planar graphs.

(a) *Every planar graph can be node colored with 6 colors in polynomial time.*

(b) *Every planar graph can be node colored with 4 colors [the famous Four Color Theorem by Appel and Haken, 1997].*

(c) *The decision problem "Is the planar graph G node-3-colorable?" is* **NP***-complete [GJ79,p. 87].*

(d) *It can be decided in polynomial time whether a graph G is node-2-colorable and, if so, such a coloring can be computed.*

The proof of the Four Color Theorem can be transformed into an algorithm for finding a node-4-coloring that has a runtime of $O(|V|^2)$ [RSST96]. An essential part of this method is a case distinction that consists of 633 cases (in the original proof by Appel and Haken this were even 1818 cases). It is therefore quite tedious to implement the algorithm. We will therefore only concentrate on a node-6-coloring. For this consider the following algorithm:

Algorithm ColPlan

1. test according to Fact 2.7(d) whether G is node-2-colorable
2. if not, color the nodes according to Fact 2.7(a) with 6 colors.

For this algorithm it obviously holds:

Theorem 2.8 ColPlan *guarantees an absolute error of $\kappa_{\text{ColPlan}}(G) \leq 3$, i.e.*

$$|\text{ColPlan}(G) - \text{OPT}(G)| \leq 3 \ .$$

Had we used in (2) the algorithm for the Four Color Theorem, we would have gotten an absolute error of 1.

2.3 Edge coloring of arbitrary graphs

Now we consider the problem of finding good edge colorings. A first approach could be again to use a greedy algorithm. Such an algorithm will be analyzed in one of the assignments. There is an interesting theorem in graph theory about edge colorings of arbitrary graphs that, in contrast to the Four Color Theorem, has a clean and easy proof [Bol98, p. 153] and provides an almost optimal edge coloring algorithm.

Theorem 2.9 (Vizing, 1964) *Every graph G needs at least $\Delta(G)$ and at most $\Delta(G)+1$ colors for an edge coloring.*

Proof. It is clear that at least $\Delta(G)$ colors are needed.

In order to prove the upper bound, we will give an algorithm that computes an edge coloring with at most $\Delta(G)+1$ colors. We say that the color c is *missing* at node u if none of the edges incident to u is colored with c. Note that at every node at least one color is missing, since we have $\Delta(G)+1$ different colors. The following observation is a key element of the proof:

Lemma 2.10 *Let G be edge colored with the colors $\{1,\ldots,\Delta(G)+1\}$, and let u and v be nodes with $\{u,v\} \notin E$ and $\deg(u),\deg(v) < \Delta(G)$. Then G can be recolored in such a way that the same color is missing at u and v.*

Proof. Let s be the color initially missing at u and c_1 be the color initially missing at v. In the following, we will take care that c_1 is also missing at u. For this we construct with the following algorithm a sequence $(v_1,\ldots,v_h)$ of neighbors of u and a sequence $(c_1,\ldots,c_{h+1})$ of colors with the property: edge $\{u,v_i\}$ is colored with c_i and c_{i+1} is missing at v_i.

```
i = 1
while there is a c_i-colored edge {u, v_i} ∈ E, v_i ∉ {v_1, ..., v_{i-1}}
do
    c_{i+1} = color missing at v_i
    i = i + 1
```

Let h be the number of times the while-loop is executed, i.e. the edge $\{u,v_h\}$ is the last that was considered. Figure 3 shows the computed sequences.

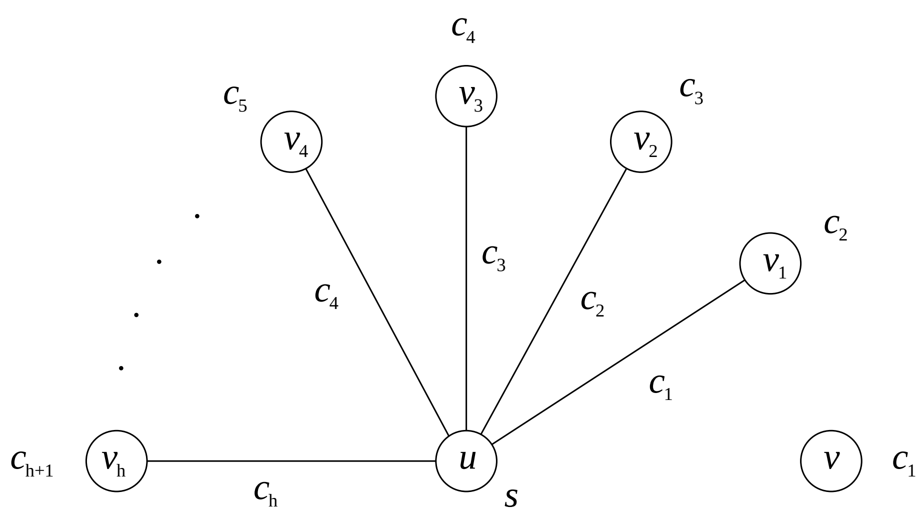

Figure 3: Determination of the v_i and c_i.

Since only neighbors of u are considered, the while loop is only executed $\Delta(G)-1$ times. There are two possibilities to end the loop:

(a) There is no edge incident to u that is colored with c_{h+1}, or

(b) there is an edge $\{u,v_j\}$ so that $c_j = c_{h+1}$.

In the case (a) we can easily "move" the colors: color all edges $\{u,v_i\}$ with the color c_{i+1}. This causes c_1 to become a missing color for u and v (cf. Figure 4).

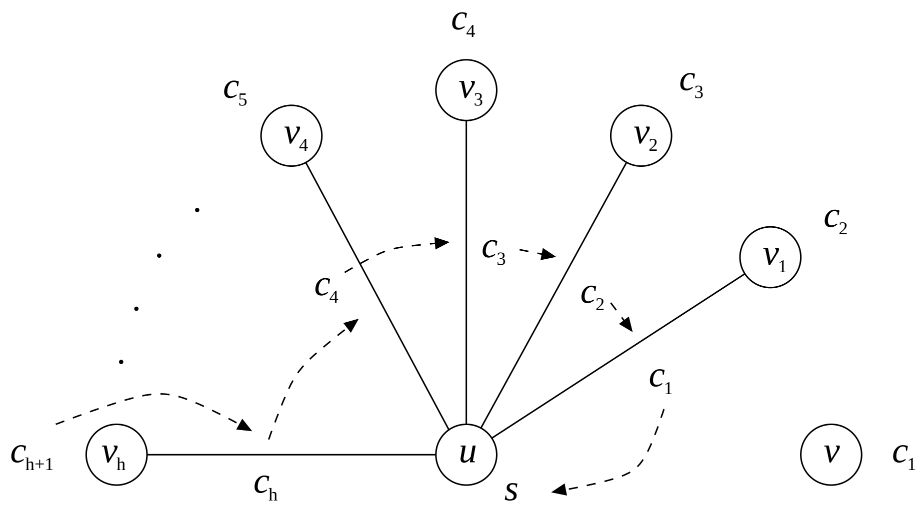

Figure 4: Treatment of case (a).

The treatment of case (b) is a bit more complex. First, we perform the color movement with the edges $\{u, v_{j-1}\}, \ldots, \{u, v_1\}$ as in case (a) and then remove the color of edge $\{u, v_j\}$ (cf. Figure 5). This causes c_1 to become a missing color at u, and we have to color now $\{u, v_j\}$.

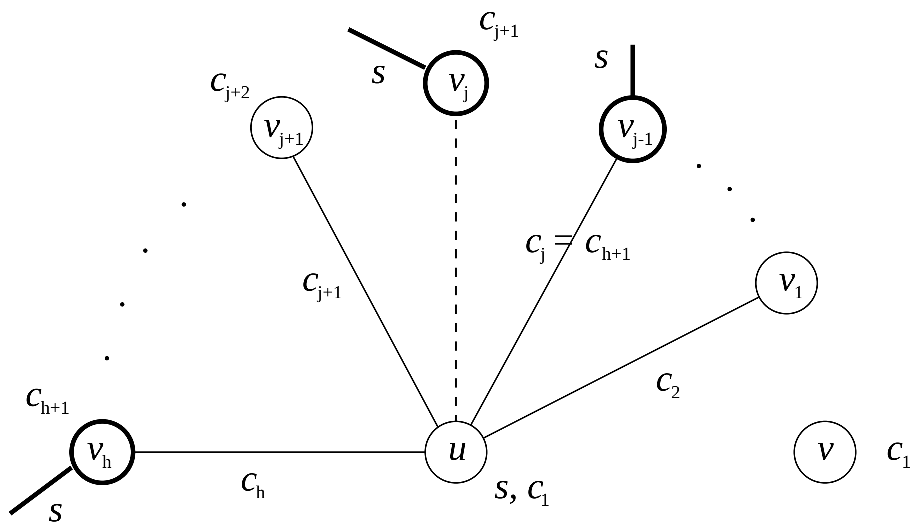

Figure 5: Case (b) after moving the colors. Possible parts of $H(s, c_{h+1})$ are made bold.

For this we consider the subgraph $H(s, c_{h+1})$ of G that consists only of edges that are colored with the colors s or c_{h+1} and all the corresponding nodes. Since every node in $H(s, c_{h+1})$ belongs to at most two edges, the connected components can only be cycles or paths. The nodes u, v_j and v_h even belong to at most one edge in $H(s, c_{h+1})$, that is, they are endpoints of paths in $H(s, c_{h+1})$. Thus, u, v_j and v_h cannot be in the same connected component in $H(s, c_{h+1})$. The edge that leaves v_j and v_h must be colored with s. We distinguish between two cases:

(b1) u and v lie in different connected components.

(b2) u and v_h lie in different connected components.

In the case (b1) we can exchange the colors s and c_{h+1} in the component of v_j in a way that now both u and v_j are missing the color s, and we can therefore color $\{u, v_j\}$ with s.

In the case (b2) we can move the colors with the edges $\{u, v_{h-1}\}, \ldots, \{u, v_j\}$ as in case (a) and can then remove the color of edge $\{u, v_h\}$. This reduces this case to case (b1) and can therefore be treated in the same way. □

Note that an important prerequisite for the lemma to work is that *every* node must have a missing color. Also note that the number of used colors can increase: s does not have to occur in the initial coloring. With the help of Lemma 2.10 we can now easily prove Theorem 2.9 by using a recursive algorithm. We take an arbitrary edge in G, color the remaining graph in a recursive way and treat the nodes of the taken edge according to Lemma 2.10.

Algorithm ColorEdges($G = (V, E)$)

```
if |E| = ∅ then return
else
    choose an arbitrary edge {u, v} ∈ E
    G' = G \ {u, v}
    ColorEdges(G')    [now G' is edge colored with ≤ Δ(G) + 1 colors]
    if a color in {1, ..., Δ(G) + 1} is missing at both u and v then
        color the edge {u, v} with the smallest color out of {1, ..., Δ(G) + 1}
        missing at u and v
    else
        recolor G' acc. to Lemma 2.10 so that some color c is missing at u and v,
        and color {u, v} with c
```

The algorithm can also be formulated in a non-recursive way so that, starting with a graph on V with empty edge set, edges in E are inserted one by one and each edge is colored according to the if statement above. □

Since there are graphs that need at least $\Delta(G) + 1$ colors for an edge coloring (for example, the complete graphs with an odd number of nodes), the theorem by Vizing cannot be improved.

Theorem 2.11 *Algorithm* ColorEdges *can be executed in time $O(|V| \cdot |E|)$ and guarantees an absolute error of 1.*

It will be an assignment to show that the absolute deviation is also 1. As already mentioned, the decision problem whether a graph G can be edge colored with $\Delta(G)$ colors is **NP**-complete [Hol81]. Holyer even shows this for a very restricted class of graphs: he shows that to answer the question "Can a 3-regular graph G be colored with 3 colors?" is **NP**-complete.

2.4 An impossibility result

Unfortunately, there are problems for which there may not be a polynomial time approximation algorithm that achieves a constant absolute error. We will now give a problem-oriented lower bound for a concrete problem Π (instead of just giving an algorithm-oriented lower bound). For this we consider Knapsack (see Example 1.7(b)).

Theorem 2.12 *If* $\mathbf{P} \neq \mathbf{NP}$*, then there is no constant* $k \in \mathbb{N}$ *so that there is a polynomial time approximation algorithm* A *for the knapsack problem with*

$$|A(I) - \mathrm{OPT}(I)| \leq k .$$

Proof. We assume on the contrary that such a k and A exist and will show that in this case it would be possible to give even an *exact* algorithm A' with a polynomial runtime for the knapsack problem. Since with such an optimal solution the **NP**-complete decision variant of the knapsack problem can be solved (see Theorem 1.8), it would follow that $\mathbf{P} = \mathbf{NP}$, which is a contradiction.

Let $I = \langle W, vol, p, C \rangle$ be an instance of the knapsack problem. Out of this we construct the instance $I' = \langle W', vol', p', C' \rangle$ with $W' = W$, $vol' = vol$, $C' = C$ and $p'(w) = (k+1)p(w)$. Obviously, a feasible solution σ for I is also a feasible solution for I' and vice versa, since vol was not changed. That is, $S(I) = S(I')$. The values of a $\sigma \in S(I) = S(I')$ are:

$$\begin{aligned} f_I(\sigma) &= \sum_{w \in \sigma} p(w) \\ f_{I'}(\sigma) &= \sum_{w \in \sigma} (k+1)p(w) = (k+1) \cdot f_I(\sigma) \end{aligned}$$

Due to the monotonicity of multiplication with $k+1$, the optimal solution σ_{opt} for I is also an optimal solution for I'.

Now we imagine the feasible solutions to be ordered in increasing order of their value, i.e. $S(I) = \{\sigma_1, \ldots, \sigma_{|S(I)|}\}$ with $f_I(\sigma_i) \leq f_I(\sigma_{i+1})$. Note that also $f_{I'}(\sigma_i) \leq f_{I'}(\sigma_{i+1})$, since due to the multiplication with a fixed value no feasible solution can overtake another feasible solution. Let j be the index with $f_I(\sigma_{j-1}) < \mathrm{OPT}(I)$ and $f_I(\sigma_j) = \mathrm{OPT}(I)$. Then $f_I(\sigma_j) - f_I(\sigma_{j-1}) \geq 1$ and

$$f_{I'}(\sigma_j) - f_{I'}(\sigma_{j-1}) = (k+1) \cdot (f_I(\sigma_j) - f_I(\sigma_{j-1})) > k .$$

That is, between the value of an optimal solution for I' and the values of all non-optimal solutions for I' is a gap of more than k.

Now we are ready to give the simple algorithm A':

1. transform instance I into I'
2. compute with the help of A a solution σ for I'
3. output $\sigma_{A'} = \sigma$

A' obviously runs in polynomial time. Furthermore, it holds that $\mathrm{OPT}(I') - f_{I'}(\sigma_{A'}) \leq k$ because of step 2. Since in I' there is no feasible solution with such a property apart from the optimal solution, $\sigma_{A'}$ must be an optimal solution for I' and therefore also for I. □

This proof contains a typical approach if one wants to show that a problem is hard to approximate: via a reduction of the original instance I of Π to an instance I' of Π' a gap is created between the optimal value and the best non-optimal value which may be small for I but is large for I'. The approach is called *scaling* or *gap amplification*.

Another important property of the proof above is the fact that we reduced the problem to itself. This approach is called *self reduction*. If we denote the problem "Approximate the knapsack problem with absolute error k" by Knapsack$\{k\}$, then we showed that Knapsack$\{0\} \leq_p$ Knapsack$\{k\}$ for all $k \in \mathbb{N}$. This means that, roughly speaking, to find an optimal solution to the knapsack problem is not harder than finding a solution with an absolute error of k.

One can view the amplification approach as if pumping up a balloon. When the balloon is pumped up, the points separate without the optimal point losing its property. If the balloon is pumped up so much that in an area of radius k around the optimum there is no other point, an approximation algorithm with absolute error at most k must output an optimal point. It is important that the pumping can be done in polynomial time. This method will be applied to other problems in the assignments.

Note that k does not even have to be a constant. In one of the assignments it will be determined how much k can grow with the size of the input so that still $\mathbf{P} = \mathbf{NP}$ would follow.

We saw that the knapsack problem cannot (unless $\mathbf{P} = \mathbf{NP}$) be approximated with a constant absolute error. However, if we only wish to achieve an additive error that is small compared to the optimal solution, then this aim is indeed achievable, as we will see in Section 4.

References

[Bol98] B. Bollobas. *Modern Graph Theory.* Springer, New York, 1998.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[Hal93] M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19–23, 1993.

[Hol81] I. Holyer. The np-completeness of edge-coloring. *SIAM Journal on Computing*, 10:718–720, 1981.

[RSST96] N. Robertson, D.P. Sanders, P. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In *Proc. of the 28th ACM Symp. on Theory of Computing (STOC)*, pages 571–575, 1996.