Polynomial Time Approximation Schemes

Hadas Shachnai

Computer Science Department

The Technion, Haifa, Israel.

E-mail: hadas@cs.technion.ac.il

Tami Tamir

School of Computer Science

The Interdisciplinary Center, Herzliya, Israel.

E-mail: tami@idc.ac.il

1 Introduction

Let Π be an NP-hard optimization problem, and let A be an approximation algorithm for Π. For an instance I of Π, let $A(I)$ denote the objective value when running A on I, and let $OPT(I)$ denote the optimal objective value. The approximation ratio of A for the instance I is $R_A(I) = A(I)/OPT(I)$, thus, when Π is minimization (maximization) problem $R_A(I) \geq 1$ ($R_A(I) \leq 1$).

A polynomial time approximation scheme is an algorithm that takes as input an additional parameter, $\varepsilon > 0$, which determines the desired approximation ratio. As ε approaches 0, the approximation ratio gets arbitrarily close to 1. The time complexity of the scheme is polynomial in the input size, but may be exponential in $1/\varepsilon$. This gives a clear trade-off between running time and quality of approximation. Formally,

Definition 1.1 *An* approximation scheme *for an optimization problem* Π *is an algorithm* A *which takes as input an instance* I *of* Π *and an error bound* ε*, runs in time polynomial in* $|I|$ *and has approximation ratio* $R_A(I, \varepsilon) \leq (1 + \varepsilon)$*. In fact, such an algorithm* A *is a family of algorithms* A_ε *such that for any instance* I*,* $R_{A_\varepsilon}(I) \leq (1 + \varepsilon)$*.*

The approximation algorithm A may be deterministic or randomized. In the latter case the result is a randomized approximation scheme.

Definition 1.2 *A* randomized approximation scheme *for an optimization problem* Π *is a family of algo-*

rithms A_ε which run in time polynomial in $|I|$ and have, for any instance I, expected *approximation ratio* $EXP[R_{A_\varepsilon}(I)] \leq (1+\varepsilon)$.

In some approximation schemes, an additive constant k, whose value is independent of I and ε, is added to the approximation ratio. Asymptotically, this constant is negligible, thus, such a scheme is called an asymptotic PTAS.

Definition 1.3 *An* asymptotic approximation scheme *for an optimization problem* Π *is a family of algorithms A_ε that run in time polynomial in $|I|$, such that, for some constant k and for any instance I, $A_\varepsilon(I) \leq (1+\varepsilon)OPT(I) + k$.*

We refer the reader to Chapter R-12 in this book for a detailed study of such schemes.

Some approximation algorithms provide a solution for a *relaxed instance* of the problem. For example, in packing problems, an algorithm may pack the items in bins whose sizes are slightly larger than the original. The objective value is achieved relative to the relaxed instance. This type of algorithm is called a *dual* approximation algorithm [23], or approximation with *resource augmentation* [8]. A *dual approximation scheme* is a family of algorithms A_ε that run in time polynomial in $|I|$, such that, for any instance I, $A(I) \leq (1+\varepsilon)OPT(I)$, and $A(I)$ is achieved for resources augmented by factor of $(1+\varepsilon)$.

Depending on the function $f(|I|, 1/\varepsilon)$ which gives the running time of the scheme, some schemes are classified as *quasi-polynomial* and others as *fully polynomial*. In particular, when the running time is $O(n^{polylog(n)})$ we get a quasi PTAS (see, e.g. [4], [13]); when the running time is polynomial in both $|I|$ and $1/\varepsilon$ we get a *fully polynomial time approximation scheme (FPTAS)*. Such schemes are studied in detail in Chapter R-11.

There is wide literature on approximation schemes for NP-hard problems. Many of these works present PTASs for certain subclasses of instances of problems, which are in general extremely hard to solve. While some of the proposed schemes may have running times which render them inefficient in practice, these works essentially help identify the class of problems that admit PTAS. There have been some studies also towards characterizing this class of problems (see, e.g. [33], [48] and Chapter R-14 in this book). We focus here on the techniques that have been repeatedly used in developing PTASs.

We refer the reader also to the comprehensive survey on Approximation Algorithms by Motwani [37], a tutorial by Schuurman and Woeginger [42], and the survey on scheduling by Karger, Stein and Wein [29],

from which we borrowed some of the examples in this chapter.

2 Partial Enumeration

2.1 Extending Partial Small-Size Solutions

There are two main techniques based on extending partial small-size solutions. The first technique exploits our ability to solve the problem *optimally* on a constant-size subset of the instance. Thus, initially, such a constant-size subset is selected. This subset contains the most 'significant' elements in the instance. We identify elements as *significant* depending on the problem at hand. The problem is solved optimally for this subset. This can be done by exhaustive search, since there is only a constant number of elements to consider. Next, this optimal partial solution is extended into a complete one, using some heuristic which has a bounded approximation ratio.

In the second technique, none of the elements is initially identified as 'significant'; instead, *all* partial solutions of constant-size are considered, and each is extended to a complete solution using some heuristic. The best extension is selected to be the output of the scheme.

The time complexity analysis of such PTASs is based on the fact that the number of possible subsets, or solutions that are considered, is exponential in the (constant) size of these subsets. The step in which the constant-size partial solution is extended is usually based on some greedy rule that may require sorting, and is polynomial. The parameter ε specifying the required approximation ratio of $(1+\varepsilon)$, determines the size k of the partial solution to which an exponential exhaustive search is applied. This implies that the running time of such schemes is exponential in $1/\varepsilon$.

2.1.1 Extending an Optimal Solution for a Single Subset

We present the first technique in the context of a classical scheduling problem, namely, the problem of finding the *minimum makespan* (or, overall completion time) of a schedule of n jobs on m identical machines. A main idea in the PTAS of Graham [21] is to schedule first optimally the k longest jobs and then schedule, using some heuristic, the remaining jobs. Formally, the input for the minimum makespan problem consists

of n jobs and m identical machines. The goal is to schedule the jobs non-preemptively on the machines in a way that minimizes the maximum completion time of any job in the schedule.

Denote by $p_1, \ldots, p_n$ the processing times of the jobs. Assume that $n > m$, and that the processing times are sorted in non-increasing order, that is, for all $i < j$, $p_i \geq p_j$. A well-known heuristic for the makespan problem is the *LPT rule*, which selects the longest unscheduled job in the sorted list and assigns it to a processor which currently has the minimum load. The PTAS combines an optimal schedule of the longest k jobs with the LPT rule, applied to the remaining jobs.

Formally, for any $k \in [0, n]$, the algorithm A_k is defined as follows:

1. Schedule optimally, with no intended idles, the first k jobs.

2. Add the remaining jobs greedily using the LPT rule.

Theorem 1.1 *Let $A_k(I)$ denote the makespan achieved by A_k on an instance I, and let $OPT(I)$ denote the minimum makespan of I, then*

$$A_k(I) \leq OPT(I)(1 + \frac{1 - \frac{1}{m}}{1 + \lfloor k/m \rfloor}).$$

Proof

Let T denote the makespan of an optimal schedule of the first k jobs. Clearly, T is a lower bound for $OPT(I)$, thus, if the makespan is not increased in the second step, i.e., $A_k(I) = T$, then A_k is optimal for I. Otherwise, the makespan of the schedule is larger than T. Let j be the job to determine the makespan (the one which completes last). By the definition of LPT, this implies that all the machines were busy when job j started its execution (otherwise, job j could start earlier). Since the optimal schedule from step 1 has no intended idles, all the machines are busy during the time interval $[0; A_k(I) - p_j]$

Let $P = \sum_{j=1}^{n} p_j$ be the total processing time of the n jobs. By the above, $P \geq m(A_k(I) - p_j) + p_j$. Also, since the jobs are sorted in nonincreasing order of processing times, we have that $p_j \leq p_{k+1}$, and therefore $P \geq mA_k(I) - (m-1)p_{k+1}$. A lower bound for the optimal solution is the makespan of a schedule in which the load on the m machines is perfectly balanced; thus, $OPT(I) \geq P/m$, which implies that $A_k(I) \leq OPT(I) + (1 - \frac{1}{m})p_{k+1}$.

In order to bound $A_k(I)$ in terms of $OPT(I)$, we need to bound p_{k+1} in terms of $OPT(I)$. To obtain

such a bound, consider the $k+1$ longest jobs. In an optimal schedule, some machine is assigned at least $\lceil (k+1)/m \rceil \geq 1 + \lfloor k/m \rfloor$ of these jobs. Since each of these jobs has processing time at least p_{k+1}, we conclude that $OPT(I) \geq (1 + \lfloor k/m \rfloor) p_{k+1}$, which implies that $p_{k+1} \leq OPT(I)/(1 + \lfloor k/m \rfloor)$. It follows that

$$A_k(I) \leq OPT(I)(1 + \frac{(1 - \frac{1}{m})}{(1 + \lfloor k/m \rfloor)}).$$

□

To observe that the above family of algorithms is a PTAS, we relate the value of k to $(1+\varepsilon)$, the required approximation ratio. Given $\varepsilon > 0$, let $k = \lceil \frac{1-\varepsilon}{\varepsilon} m \rceil$. It is easy to verify that the corresponding algorithm A_k achieves approximation ratio at most $(1+\varepsilon)$. Thus, we conclude that for a fixed m, there is a polynomial time approximation scheme for the minimum makespan problem.

Note that for any fixed k, an optimal schedule of the first k jobs can be found in $O(m^k)$ steps. Applying the LPT rule takes additional $O(nlogn)$. For A_ε, we get that the running time of the scheme is $O(m^{m/\varepsilon})$, i.e., exponential in m (that is assumed to be constant) and $1/\varepsilon$. This demonstrates the basic property of approximation schemes: a clear trade-off between running time and the quality of approximation.

2.1.2 Extend All Possible Solutions for Small Subsets

The second technique, of considering all possible subsets, is illustrated in an early PTAS of Sahni for the *knapsack problem* [40]. An instance of the knapsack problem consists of n items, each having a specified size and a profit, and a single knapsack, having size B. Denote by $s_i \geq 0, p_i \geq 0$ the size and profit associated with item i. The goal is to find a subset of the items such that the total size of the subset does not exceed the knapsack capacity, and the total profit associated with the items is maximized.

The PTAS in [40] is based on considering all $O(kn^k)$ possible subsets of size at most k, where k is some fixed constant. Each of these subsets is extended to a larger feasible subset by adding more items to the knapsack, using some greedy rule. The best extension among these $O(kn^k)$ candidates is selected to be the output of the scheme. Formally, for any $k \in [0, n]$, the algorithm A_k is defined as follows:

1. (Preprocessing) Sort the items in non increasing order of their profit densities, p_i/s_i.

2. For each feasible subset of at most k items,

(a) Pack the subset in the knapsack.

(b) Add to the knapsack items in the sorted list one by one, while there is enough available capacity.

3. Select among the packings generated in Step 2. one which maximizes the profit.

Theorem 1.2 *Let $P(A_k)$ denote the profit achieved by A_k, and let $P(OPT)$ denote the optimal profit, then*

$$OPT(A_k) \leq P(A)(1 + \frac{1}{k}).$$

Proof

Let OPT be any optimal solution. If $|OPT| \leq k$ we are done, since the subset OPT will be considered in some iteration of Step 2. Otherwise, let $H = \{a_1, a_2, \ldots, a_k\}$ be the set of k most profitable items in OPT. There exists an iteration of A_k in which H is considered. We show that the profit gained by A_k in this iteration yields the statement of the theorem. Consider the list $L_1 = OPT \setminus H = \{a_{k+1}, \ldots, a_x\}$ of the remaining items of OPT, in the order they are considered by A_k. Recall that, at some point, A_k will try H as the initial set of k packed items. The algorithm will then add greedily items, as long as the capacity constraint allows. If all the items are packed, A_k is clearly optimal; otherwise, at some point there is not enough space for the next item. Let m be the index of the first item in L_1 which is not packed in the knapsack by A_k, i.e., the items $a_{k+1}, \ldots, a_{m-1}$ are packed. The item a_m is not packed because B_e, the remaining empty space at this point, is smaller than s_m. The greedy algorithm packed into the knapsack only items with profit density at least p_m/s_m. At the time that a_m is dropped, the knapsack contains the items from H, the items $a_{k+1}, \ldots, a_{m-1}$ and some items which are not in OPT.

Let G denote the items packed in the knapsack so far by the greedy stage of A_k. All of these items have profit density at least p_m/s_m. In particular, the items in $G \setminus OPT$ that have total size $\Delta = B - (B_e + \sum_{i=1}^{m-1} s_i)$ all have profit density at least p_m/s_m. Thus, the total profit of the items in G is $P(G) \geq \sum_{i=k+1}^{m-1} p_i + \Delta \frac{p_m}{s_m}$. We conclude that the total profit of the items in OPT is

$$\begin{aligned} P(OPT) &= \sum_{i=1}^{k} p_i + \sum_{i=k+1}^{m-1} p_i + \sum_{i=m}^{|OPT|} p_i \\ &\leq P(H) + (P(G) - \Delta \frac{p_m}{s_m}) + (B - \sum_{i=1}^{m-1} s_i) \frac{p_m}{s_m} \\ &= P(H) + P(G) + B_e \frac{p_m}{s_m} < P(H \cup G) + p_m \end{aligned}$$

Since A_k packs at least $H \cup G$, we get that $P(A_k) \geq P(H)+P(G)$, which implies that $P(OPT)-P(A_k) < p_m$. Given that there are at least k items with a profit at least as large as a_m (those selected to H), we conclude that $s_m \leq S(OPT)/(k+1)$. This gives the approximation ratio. □

Assuming a single preprocessing step, in which the items are sorted by their profit densities, each subset is extended to a maximal packing in time $O(n)$. Since there are $O(kn^k)$ possible subsets to consider, the total running time of the scheme is $O(kn^{k+1})$.

To obtain a PTAS for the knapsack problem, let A_ε be the algorithm A_k with $k = \lceil 1/\varepsilon \rceil$. By the above, the approximation ratio is at most $1+\varepsilon$, and the running time of A_ε is $O(\frac{1}{\varepsilon} n^{1+\frac{1}{\varepsilon}})$.

The technique of choosing the best among a small number of partial packings was applied also to variants of multidimensional packing. A detailed example is given in Section 3.2.

2.2 Applying Enumeration to a Compacted Instance

In this section we present the technique of applying exhaustive enumeration to a modified instance, in which we have a more compact representation of the input. Approximation schemes that are based on this approach consist of three steps:

1. The instance I is modified to a simpler instance, I'. The parameter ε determines how rough I' is compared to I. The smaller ε the more refined is I'.

2. The problem is solved optimally on I'.

3. An approximate solution for I is induced from the optimal solution for I'.

The challenge is to modify I in the first step into an instance I' that is simple enough to be solved in polynomial time, yet not too different from the original I, so that we can use an exact solution for I' to derive an approximate solution for I.

The use of this technique usually involves partitioning the input into *significant* and *non-significant* elements. The partition depends on the problem at hand. For example, it is natural to distinguish between *long* and *short* jobs in scheduling problems, and between *big* and *small*, or *high-profit* and *low-profit* elements, in packing problems. For a given instance, the distinction between the two types of elements usually depends

on the input parameters (including ε), and on the optimal solution value.

In some cases, the transformation from I to I' involves only grouping the non-significant elements. Each group of such elements thus forms a single significant element in I'. As a result, the instance I' consists of a small number of significant elements. More details and an example for this type of transformation are given in Section 2.2.1.

In other cases, all the elements, or only the more significant ones, are transformed into a set of elements with a small number of distinct values. This approach is described and demonstrated in Section 2.2.2.

2.2.1 Grouping Subsets of Elements

We illustrate the technique with the PTAS of Sahni [41] for the minimum makespan problem on two identical machines. The input consists of n jobs with processing times $p_1, \ldots, p_n$. The goal is to schedule the jobs on two identical parallel machines in a way that minimizes the latest completion time. In other words, we seek a schedule which balances the load on the two machines as much as possible.

Let $P = \sum_{j=1}^{n} p_j$ denote the total processing time of all jobs, and let p_{max} denote the longest processing time of a job. Let $C = \max(P/2, p_{max})$. Note that C is a lower bound on the minimum makespan (that is, $OPT \geq C$), since $P/2$ is the schedule length if the load is perfectly balanced between the two machines, and since some machine must process the longest job.

The first step of the scheme is to modify the instance I into a simplified instance I'. This modification depends on the value of C and on the parameter ε. Given I, ε, partition the jobs into small jobs – of length at most εC, and big jobs – of length larger than εC. Let P_S denote the total length of small jobs. The modified instance I' consists of the big jobs in I together with $\lfloor P_S/(\varepsilon C) \rfloor$ jobs of length εC.

Next, we need to solve optimally the minimum makespan problem for the instance I'. Note that all jobs in I' have length at least εC and their total size is at most P, the total processing time of the jobs in the original instance, since the small jobs in I are replaced in I' by jobs of length εC with total length at most P_S. Therefore, the number of jobs in I' is at most *the constant* $P/\varepsilon C \leq 2/\varepsilon$. An optimal schedule of a constant number of jobs can be found by exhaustive search over all $O(2^{2/\varepsilon})$ possible schedules. This constant number is independent of n, but grows exponentially with ε, as we expect from our PTAS.

Finally, we need to transform the optimal schedule of I' into a feasible schedule of I. Note that, for the makespan objective, we are only concerned about the partition of the jobs between the machines, while the order in which the jobs are scheduled on each machine can be arbitrary. Denote by $OPT(I')$ the length of the optimal schedule for I'. To obtain a schedule of I, each of the big jobs is scheduled on the same machine as in the optimal schedule for I'. The small jobs are scheduled greedily in an arbitrary order on the first machine until, for the first time, the total load on the first machine is at least $OPT(I')$. The remaining small jobs are scheduled on the second machine. Clearly, the overflow on the first machine is at most εC (maximal length of a small job). Also, since the total number of (εC)-jobs was defined to be $\lfloor P_S/(\varepsilon C) \rfloor$, the overflow on the second machine is also bounded by εC. Therefore, the resulting makespan in the schedule of I is at most $OPT(I') + \varepsilon C$.

To complete the analysis we need to relate $OPT(I')$ to $OPT(I)$.

Claim 1.3 $OPT(I') \leq (1+\varepsilon)OPT(I)$

Proof

Given a schedule of I, in particular an optimal one, a schedule for I' can be derived by replacing – on each machine separately – the small jobs with jobs of size εC, with at least the same total size. Recall that the number of (εC)-jobs in I' is $\lfloor P_S/(\varepsilon C) \rfloor$. Regardless of the partition of the small jobs in I between the two machines, the result of this replacement is a feasible schedule of I' whose makespan is at most $OPT(I)+\varepsilon C$. Since $OPT(I) \geq C$, the statement of the claim holds. □

Back to our scheme, we showed that the optimal schedule of I' is transformed into a feasible schedule of I whose makespan is at most $OPT(I') + \varepsilon C$. By Claim 1.3, this value is at most $(1+\varepsilon)OPT(I) + \varepsilon C \leq (1+2\varepsilon)OPT(I)$. By selecting $\varepsilon' = \varepsilon/2$, and running the scheme with ε', we get the desired ratio of $(1+\varepsilon)$.

The above scheme can be extended to any constant number of machines. For *arbitrary* number of machines, a more complex PTAS exists: the scheme of [23], which requires reducing the number of distinct values in the input, is given in the next section.

2.2.2 Reducing the Number of Distinct Values in the Input

Any optimization problem can be solved optimally in polynomial, or even constant time, if the input size is some constant. For many optimization problems, an efficient algorithm exists if the input size is arbitrary but the number of *distinct values* in the input is some constant. Alternatively, the problem can be solved by a pseudo-polynomial-time algorithm (e.g., by dynamic programming), whose running time depends on the instance parameters, and is therefore polynomial only if the parameter values are polynomial in the problem size.

The idea behind the technique that we describe below is to transform the elements (or sometimes, only the significant elements) in the instance I into an instance I' in which the number of distinct values is fixed, or to scale the values according to the input size. The problem is then solved on I', and the solution for I' is transformed into a solution for the original instance. The non-significant elements, which are sometimes omitted from I', are added later to the solution, using some heuristic. The parameter ε determines the (constant) number of distinct values contained in I': the smaller ε, the larger is the number of distinct values. The following are two main approaches for determining the values in I'.

1. **Rounding** – The values in I' form an arithmetic series in which the difference between elements is a function of ε. For example, multiples of $\varepsilon^2 T$, for some value T. In this approach, the gap between any two values bounds the difference between the original value of an element in I and the value of the corresponding element in I'. Note that the number of elements whose values are rounded to a single value in I' can be arbitrary.

2. **Shifting** – The values in I' are a subset of the values in I, selected such that the distribution on the number of values in I that are shifted to a single value in I' is uniform. On the other hand, in contrast to the rounding approach, there is no bound on the difference between the value of an element in I and its value in I'. For example, partition the elements into $\lceil 1/\varepsilon^2 \rceil$ groups, each having at most $\lfloor n/\varepsilon^2 \rfloor$ elements, and fix the values in each group to be (say) the minimal value of any element in the group.

In both approaches, the approximation ratio is guaranteed to be $(1+\varepsilon)$ if I' is close enough to I. Formally, an optimal solution for I' induces a solution for I whose value is larger/smaller by a factor of at most $(1+\varepsilon)$.

Another factor of $(1+\varepsilon)$ may be added to the approximation ratio due to the non-significant items – in case they are handled separately.

We demonstrate this technique with the classic PTAS of Hochbaum and Shmoys [23] for the minimum makespan problem on parallel machines. The input for the problem is a set of n jobs having processing times $p_1, \ldots, p_n$, and m identical machines; the goal is to schedule the jobs on the machines in a way that minimizes the latest completion time of any job. The number of machines, m, can be arbitrarily large (otherwise, a simpler PTAS exists; see Section 2.2.1).

First, note that the *minimum makespan* (MM) problem is closely related to the *bin packing* (BP) problem. The input for BP is a collection of items whose sizes are in $[0, 1]$. The goal is to pack all items using a minimal number of bins. Formally, let $I = \{p_1 \ldots p_n\}$ be the sizes in a set of n items, where $0 \leq p_j \leq 1$. The goal is to find a collection of subsets $U = \{B_1, B_2, \ldots, B_k\}$ which forms a disjoint partition of I, such that for all i, $1 \leq i \leq k$, $\sum_{j \in B_i} p_j \leq 1$, and the number of bins, k, is minimized.

The exact solutions of MM and BP relate in the following way. It is possible to schedule all the jobs in an MM instance on m machines with makespan C_{max} if and only if it is possible to pack all the items in a BP instance, where the size of item j is p_j/C_{max}, in m bins. The relation between the optimal solutions does not remain valid for approximations. In particular, BP admits an asymptotic FPTAS (see chapter R-12) while MM does not. However, this relation can be used to develop a PTAS for MM.

Let $OPT_{BP}(I)$ be the number of bins in an optimal solution of BP, and let $OPT_{MM}(I) = C_{max}$ be an optimal solution for MM. Denote by $\frac{I}{d}$ the BP input in which all the values are divided by d. We already argued that:

$$OPT_{BP}(\frac{I}{d}) \leq m \quad \Leftrightarrow \quad OPT_{MM}(I, m) \leq d$$

We define a *dual* approximation scheme for BP. For an input I, we seek a solution with at most OPT_{BP} bins, where each bin is filled to capacity at most $1+\varepsilon$. In other words, we relax the bin capacity constraint by a factor of $1+\varepsilon$. Let $dual_\varepsilon(I)$ be such an algorithm, and let $DUAL_\varepsilon(I)$ be the number of bins in the corresponding packing.

Theorem 1.4 *If there exists a dual approximation algorithm for BP, then there is a PTAS for the minimum makespan problem.*

Proof

The PTAS performs a binary search to find OPT_{MM}. In order to bound the range in which the optimal makespan is searched, two lower bounds and one upper bound for this value are used. The lower bounds are the length of the longest job, and the load on each machine when the total load is perfectly balanced. That is, let $SIZE(I,m) = max\{\frac{1}{m}\sum p_i, p_{max}\}$, then $OPT_{MM} \geq SIZE(I,m)$. The upper bound uses the fact that the simple *List Scheduling* algorithm attains a 2-approximation ratio [21], therefore $OPT_{MM} \leq 2SIZE(I,m)$.

Now it is possible to perform a binary search to find OPT_{MM}. Instead of checking whether $OPT_{MM} < d$, the algorithm checks whether $DUAL_\varepsilon(\frac{I}{d}) < m$.

$upper = 2SIZE(I,m)$

$lower = SIZE(I,m)$

repeat until $lower = upper$

    $d = (lower + upper)/2$

    call $dual_\varepsilon(\frac{I}{d})$

    if $DUAL_\varepsilon(\frac{I}{d}) > m$

        $lower \leftarrow d$

    else

        $upper \leftarrow d$

$d^\star \leftarrow upper$

return $dual_\varepsilon(\frac{I}{d^\star})$

Initially, $OPT_{MM}(I,m) \leq upper \Rightarrow OPT_{BP}(\frac{I}{upper}) \leq m$. Since $dual_\varepsilon$ is a relaxation of BP, $DUAL_\varepsilon(\frac{I}{upper}) \leq OPT_{BP}(\frac{I}{upper})$. This implies that $DUAL_\varepsilon(\frac{I}{upper}) \leq m$. By the update rule, the above remains true during the execution of the loop. However,

$$DUAL_\varepsilon(\frac{I}{upper}) \leq m \Rightarrow OPT_{MM}(I,m) \leq (1+\varepsilon)upper,$$

and thus $(1+\varepsilon)upper$ remains an upper bound on $OPT_{MM}(I,m)$ during the search. Similarly, before the loop, $OPT_{MM}(I,m) \geq lower$, which remains true since $DUAL_\varepsilon(\frac{I}{lower}) \geq m$ is an invariant of the loop, and

$$OPT_{BP}(\frac{I}{lower}) \geq DUAL_\varepsilon(\frac{I}{lower}) \geq m \Rightarrow OPT_{MM}(I,m) \geq lower.$$

Thus, the solution value is bounded above by

$$(1+\varepsilon)\cdot d^{\star} = (1+\varepsilon)\cdot upper = (1+\varepsilon)\cdot lower \leq (1+\varepsilon)OPT_{MM}(I,m)$$

In practice, assume that we stop the binary search after k iterations. At this time, it is guaranteed that $upper - lower \leq 2^{-k}SIZE(I,m) \leq 2^{-k}OPT_{MM}(I,m)$, and the value of the solution is bounded above by $(1+\varepsilon)\cdot d^{\star} = (1+\varepsilon)\cdot upper \leq (1+\varepsilon)\cdot(lower + 2^{-k}OPT_{MM}(I,m)) \leq (1+\varepsilon)(1+2^{-k})OPT_{MM}(I,m)$.

By choosing $k = O(\log\frac{1}{\epsilon})$, and taking in the scheme $\varepsilon' = \varepsilon/3$, we obtain a $(1+\varepsilon)$-approximation. □

We now describe the $dual_{\varepsilon}$ approximation scheme for bin packing. This scheme uses the rounding and grouping technique.

Theorem 1.5 *There exists an $O\left(n^{\lceil\frac{1}{\varepsilon^2}\rceil}\right)$-time dual approximation scheme for bin packing.*

Proof

Recall that, for a given $\varepsilon > 0$, the dual approximation scheme needs to find a packing of all items using at most OPT_{BP} bins, such that the total size of the items packed in each bin is at most $1+\varepsilon$. The basic idea is to omit first the "small" items and then round the sizes of the "big" items; this yields an instance in which the number of distinct item sizes is fixed. We can now solve the problem exactly using dynamic programming, and the solution induces a solution for the original instance, where each bin is filled up to capacity $1+\varepsilon$.

The first observation is that small items, whose sizes are less that ε, can be initially omitted. The problem will be solved for big items only and the small items will be added later on greedily, in the following manner: if there is a bin filled with items of total size less than 1, small items are added to it; otherwise, a new bin is opened. If no new bin is opened, then, clearly, no more than the optimum number of bins is used (as the dual PTAS uses the minimal number of bins for the big items). If new bins were added, then all original bins are filled to capacity at least 1, and all the new bins (except maybe the last one) are also filled to capacity at least 1. This is optimal since $OPT(I) \geq \lceil\sum p_i\rceil \geq DUAL_{\varepsilon}(I)$. We conclude that, without loss of generality, all items are of size $\varepsilon \leq p_i \leq 1$. Divide the range $[\varepsilon, 1]$ into intervals of size ε^2. This gives $S = \lceil\frac{1}{\varepsilon^2}\rceil$ intervals. Denote by l_i the endpoints of the intervals and let b_i be the number of elements whose sizes are in the interval $(l_i, l_{i+1}]$.

We now examine a packed bin. Since the minimal item size is ε, the bin can contain at most $\lfloor \frac{1}{\epsilon} \rfloor$ items. Denote by X_i the number of items in the bin whose sizes are in the interval $(l_i, l_{i+1}]$. X_i is in the range $[0, \lfloor \frac{1}{\epsilon} \rfloor)$. Let the vector $(X_1, \ldots, X_S)$ denote the *configuration* of the bin. The number of feasible configurations is bounded above by $\lfloor \frac{1}{\epsilon} \rfloor^S$. A configuration is *feasible* if and only if $\sum_{i=1}^{S} X_i l_i \leq 1$.

For any bin B whose packing forms a feasible configuration, the total size of the items in the bin is bounded by

$$\sum_{j \in B} p_j \leq \sum_{j \in B} X_j l_{j+1} \leq \sum_{j \in B} X_j (l_j + \varepsilon^2) \leq 1 + \varepsilon^2 \sum_{j \in B} X_j \leq 1 + \varepsilon^2 \cdot \frac{1}{\epsilon} \leq 1 + \varepsilon.$$

Therefore, it is sufficient to solve the instance with all item sizes rounded down to sizes in $\{l_1, \ldots, l_S\}$.

Finally, we describe a dynamic programming algorithm which solves the bin packing problem exactly when the number of distinct item sizes is fixed. Let $BINS(b_1, b_2, \ldots, b_S)$ be the minimal number of bins required to pack b_1 items of size l_1, b_2 items of size l_2 , ..., and b_S items of size l_S. Let $\mathcal{C}$ denote the set of all feasible configurations. Observe that, by a standard dynamic programming recursion,

$$BINS(b_1, b_2, \ldots, b_S) = 1 + min_{\mathcal{C}} BINS(b_1 - X_1, b_2 - X_2, \ldots, b_S - X_S).$$

We minimize over all possible vectors $(X_1, X_2, \ldots, X_s)$ that correspond to a feasible packing of the "first" bin (counted by the constant 1), and the best way to pack the remaining items (this is the recursive call). Thus, the dynamic programming procedure builds a table of size n^S, where the calculation of each entry requires $O(\lfloor \frac{1}{\epsilon} \rfloor^S)$.

This yields a running time of

$$O\left(n^S \cdot \lfloor \frac{1}{\epsilon} \rfloor^S\right) = O\left((\frac{n}{\varepsilon})^{\lceil \frac{1}{\varepsilon^2} \rceil}\right) = O\left(n^{\lceil \frac{1}{\varepsilon^2} \rceil}\right)$$

□

The technique of applying enumeration to a compacted instance through grouping/rounding has been extensively used in PTASs for scheduling problems (see, e.g., [43], [28], [1]). A common approach for compacting the instance is to reduce the input parameters to *poly-bounded*, i.e., parameters whose values can be bounded as function of the input size. This approach is used, e.g. in the PTAS of Chekuri and Khanna for preemptive weighted flow time [13] (see also the survey paper [29]).

2.3 More on Grouping and Shifting

In the following we outline two extensions of the techniques described in this section.

Randomized Grouping: In some cases, we need to define a partition of the input elements to groups $(I_1, \ldots, I_k)$, using for each element x a parameter of the problem, $q(x)$, such that the elements in two groups I_j and I_{j+1} differ in their $q(x)$ value by roughly a factor of α, for some $\alpha > 1$. When such partition is infeasible, we can use randomization to achieve an *expected* separation between groups. For a parameter $\alpha > 1$, the following randomized geometric grouping technique yields an expected separation that is logarithmic in α. This technique extends the deterministic geometric rounding technique described in Section 2.2.2. Initially, pick a number $r \in [1, \alpha]$ at random, by a probability distribution having the density function $f(y) = 1/y \ln \alpha$. An element x with the value $q(x)$ belongs to the group I_j if $q(x) \in [r\alpha^j, r\alpha^{j+1})$. Thus, the index of the group to which x belongs, denoted $g(x)$, is a random variable which can take two possible values: $\lfloor \log_\alpha q(x) \rfloor$ or $\lfloor \log_\alpha q(x) \rfloor + 1$. It can be shown that for a fixed α, the number of distinct partitions induced by the random choices of r is at most the number of elements in the input. This enables to easily derandomize algorithms that use randomized geometric grouping. The technique was applied, e.g., by Chekuri and Khanna [13] in a PTAS for preemptive weighted flow time.

Oblivious Shifting: While applying the standard shifting technique (as described in Section 2.2.2) requires knowing the initial input parameters, it is possible to apply shifting also when not all values are known a-priori. In *oblivious shifting*, the input size is initially known, and the scheme starts by defining the number of values in the resulting instance, but the actual shifted values are revealed at a later stage, by optimizing on these values, considering the constraints of the problem. The technique can be used for defining a 'good' compacted instance from a partial solution for the problem, which can then enable to obtain a complete solution for the problem efficiently.

For example, a variant of the bin packing problem, in which items may be *fragmented*, is solved in [46] in two steps. Given the input, we need to determine the set of items that will be fragmented, as well as the fragment sizes in a feasible approximate solution. Since the possible number of fragment sizes is large, a compact vector of fragments is generated, which contains a bounded number of *unknown* shifted fragment

sizes. The actual sizes of the shifted fragments are determined by solving a linear program which attempts to find a feasible packing of these fragments. A detailed description is given in [46].

3 Rounding Linear Programs

In this section we discuss approximations obtained using linear programming relaxation of the integer program formulation of a given optimization problem. We refer the reader to Chapters R-5 and R-94 in this book for further background on linear programming and rounding linear programs. Most generally, the technique is based on solving a linear programming relaxation of the problem, for which an exact or approximate solution can be obtained efficiently. This solution is then rounded, thus yielding an approximate integral solution. The (fractional) solution obtained for the LP needs to have some nice properties that would allow rounding to be not too harmful, in terms of ε, the accuracy parameter of the scheme. One such property of a linear program, which is commonly used, is the existence of a small *basic solution*. We illustrate below the usage of this property, with examples from vector scheduling and covering integer programs. A linear program has a *small* basic solution, if there exists an optimal solution in which the number of non-zero variables is small as function of the input size and ε. For such a solution, the error incurred by rounding can be bounded, such that the resulting integral solution is within factor of $1+\varepsilon$ from the optimal. A natural example is the class of linear programs in which either the number of variables or the number of constraints is some fixed constant. For such programs, there exists a basic solution in which the number of non-zero variables is fixed; however, depending on the problem, and in particular, on the value of an optimal solution for the LP, a basic solution can be 'small', even if the number of non-zero variables is relatively large, for example, $\Omega(\varepsilon n)$, where n is the number of variables.

LP rounding can be combined with the techniques described in Section 2. In Section 3.1 we show the usage of LP rounding for a given subset of input elements satisfying certain properties. In Section 3.2 we show how LP rounding can be combined with the selection of *all* possible (small) subsets.

3.1 Solving LP for a Subset of Elements

As mentioned in Section 2.1, in many problems, an approximation scheme can be obtained by partitioning a set of input elements to subsets, and solving the problem for each subset separately. For some subsets, a good solution can be obtained by rounding an LP relaxation of the problem.

In certain assignment problems, we can find an almost integral basic solution for an LP, for part of the input, since the relation between the number of variables and non-trivial constraints in the linear programming relaxation, combined with the assignment requirement of the problem, imply that only few variables can get fractional values. This essential property is used, e.g. in the PTAS of Chekuri and Khanna for the *vector scheduling (VS)* problem [11]. The VS problem is to schedule d-dimensional jobs on m identical machines, such that the maximum load over all dimensions and over all machines is minimized. Formally, an instance I of VS consists of n jobs, $J_1, \ldots, J_n$, where J_j is associated with a rational d-dimensional vector $(p_j^1, \ldots, p_j^d)$, and m machines. We need to assign the jobs to the machines, i.e., schedule a subset of the jobs, A_i, on machine i, $1 \leq i \leq m$, such that $\max_{1 \leq i \leq m} \max_{1 \leq h \leq d} \sum_{J_j \in A_i} p_j^h$ is minimized.

Note that in the special case where $d = 1$ we get the minimum makespan problem (see in Section 2.2.2). The PTAS of [11] for the vector scheduling problem, where d is fixed, applies a non-trivial generalization of the PTAS of Hochbaum and Shmoys for the case $d = 1$ [23]. The scheme is based on a primal-dual approach, in which the primal problem is VS and the dual problem is vector packing. Thus, the machines are viewed as d-dimensional bins, and the schedule length — as bin capacity (or height). W.l.o.g., we may assume that the optimal schedule has the value 1. Given an $\varepsilon > 0$ and a correct guess of the optimal value, we describe below an algorithm A_ε that returns a schedule of height at most $1 + \varepsilon$. Arriving at correct guess involves a binary search for the optimal value (which can be done in polynomial time; see below).

Let $\delta = \varepsilon/d$ be a parameter. The scheme starts with a preprocessing step, which enables to bound the ratio of the largest coordinate to the smallest non-zero coordinate in any input vector. Specifically, let $\| J_j \|_\infty = \max_{1 \leq h \leq d} p_j^h$ be the ℓ_∞ norm of J_j, $1 \leq j \leq n$, then, for any J_j, and any $1 \leq h \leq d$, if $p_j^h \leq \delta \| J_j \|_\infty$, we set $p_j^h = 0$. As shown in [11], any valid schedule for the resulting modified instance, I', yields a valid solution for the original instance, I, whose height is at most $(1 + \varepsilon)$ times that of I'.

We consider from now on only transformed instances. The scheme proceeds by partitioning the jobs to

the sets L (*large*) and S (*small*). The set L consists of all vectors whose ℓ_∞ norm is greater than δ, and S contains the remaining vectors. The algorithm A_ε packs first the large jobs and then the small jobs. Note, that while in the case of $d = 1$ these packings are done independently, for $d \geq 2$, we need to consider the interaction between these two sets. Similar to the scheme of Hochbaum and Shmoys [23], a valid schedule is found for the jobs by guessing a configuration. In particular, let the d-tuple $(a_1, \ldots, a_d)$ $0 \leq a_h \leq \lceil 1/\varepsilon \rceil$, $1 \leq h \leq d$, denote a *capacity configuration*, that is, the way some bin is filled. Since $d \geq 2$ is a constant, the possible number of capacity configurations, given by $W = (1 + \lceil 1/\varepsilon \rceil)^d$, is also a constant. Then, by numbering the capacity configurations, we describe by a W-tuple $M = (m_1, \ldots, m_W)$ the number of bins having capacity configuration w, where $1 \leq w \leq W$. The possible number of *bin configurations* is then $O(m^W)$. This allows to guess a bin configuration which yields the desired $(1 + \varepsilon)$-approximate solution in polynomial time.

We say that a packing of vectors in a bin *respects* a capacity configuration $(a_1, \ldots, a_d)$ if the height of the packing is smaller than εa_h for any $1 \leq h \leq d$. Given a capacity configuration $(a_1, \ldots, a_d)$, we define the *empty capacity configuration* to be the d-tuple $(\bar{a}_1, \ldots, \bar{a}_d)$, where $\bar{a}_h = \lceil 1/\varepsilon \rceil + 1 - a_h$, for $1 \leq h \leq d$. For a given bin configuration, M, we denote by $\bar{M}$ the bin configuration obtained by taking for each of the bins in M the corresponding empty capacity configuration.

The scheme performs the following two steps for each possible bin configuration, M: (i) decides whether vectors in L can be packed respecting M, and (ii) decides whether vectors in S can be packed respecting $\bar{M}$. Given that we have guessed the correct bin configuration M, both steps will succeed, and we get a packing of height at most $1 + \varepsilon$.

We now describe how the scheme packs the large and the small vectors. The vectors in L are packed using rounding and dynamic programming. In particular, since by definition, any entry in a vector in L has the value δ^2 or larger, we use geometric rounding, that is, for each vector J_j, and any entry p_j^h, $1 \leq h \leq d$, p_j^h is rounded down to the nearest value of the form $\delta^2(1 + \varepsilon)^t$, for $0 \leq t \leq \lceil \frac{2}{\varepsilon} \log 1/\delta \rceil$. Denote the resulting set of vectors L', and the modified instance I'. The vectors in L' can be partitioned into

$$q = (1 + \lceil \frac{2}{\varepsilon} \log 1/\delta \rceil)^d \tag{1.1}$$

classes. The proofs of the next lemmas are given in [11].

Lemma 1.1 *Given a solution for I', replacing each vector in L' by the corresponding vector in L results in a valid solution for I whose height is at most $1+\varepsilon$ times that of I'.*

Lemma 1.2 *Given a correct guess of a bin configuration M, there exists an algorithm which finds a packing of the vectors in L' that respects M, and whose running time is $O((d/\delta)^q mn^q)$, where q is given in (1.1).*

The small vectors are packed using a linear programming relaxation and careful rounding. Renumber the vectors in S by $1,\ldots,|S|$. Let $x_{ji} \in \{0,1\}$ be an indicator variable for the assignment of the vector J_j to machine i, $1 \le j \le n, 1 \le i \le m$. In the LP relaxation $x_{ji} \ge 0$. We solve the following linear program.

$$(LP) \qquad \sum_{J_j \in S} p_j^h x_{ji} \le b_i^h \quad 1 \le i \le m, \quad 1 \le h \le d \tag{1.2}$$

$$\textstyle\sum_{i=1}^{m} x_{ji} = 1 \quad 1 \le j \le |S| \tag{1.3}$$

$$x_{ji} \ge 0 \qquad 1 \le j \le n, \quad 1 \le i \le m \tag{1.4}$$

The constraints (1.2) guarantee that the packing does not exceed a given height bound in any dimension (i.e., the available height after packing the large vectors). The constraints (1.3) reflect the requirement that each vector is assigned to one machine. A key property of the LP, which enables to obtain an integral solution that is close to the fractional, is given in the next result.

Lemma 1.3 *In any basic feasible solution for LP, at most $d \cdot m$ vectors are assigned (fractionally) to more than one machine.*

Proof

Recall that the number of non-zero variables, in any *basic* solution for a linear program, is bounded by the number of tight constraints in some optimal solution (since non-tight constraints can be omitted). Since the number of non-trivial constraints (i.e., constraints other than $x_{ji} \ge 0$) is $(|S| + d \cdot m)$, it follows that the number of strictly positive variables in any basic solution is at most $(|S| + d \cdot m)$. Since each vector is assigned to at least one machine, the number of vectors which are fractionally assigned to more than one machine is at most $d \cdot m$. □

The above type of argument was first made and exploited by Potts [39] in the context of parallel machine scheduling. It was later applied for other problems, such as job shop scheduling (see e.g., [27]).

Thus, we solve the above program and obtain a basic solution. Denote by S' the set of vectors which are assigned fractionally to two machines or more. Since $|S'| \leq d \cdot m$, we can partition the set S' to subsets of size at most d each, and schedule the i-th set to the i-th machine. Since $\| J_j \|_\infty \leq \delta = \varepsilon/d$, for all $J_j \in S'$, the total height of the machines is violated at most by ε in any dimension. We can therefore summarize in the following theorem.

Theorem 1.6 *For any $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm for VS whose running time is $(nd/\varepsilon)^{O(f)}$, where $f = O((\frac{\ln(d/\varepsilon)}{\varepsilon})^d)$.*

Proof

By the above discussion, given the correct guess of the optimal value, the scheme yields a schedule of value (height) at most $1 + O(\varepsilon)$ the optimal. We need to find a packing of the vectors in L and S, for each bin configuration M. The running time for a single configuration is dominated by the packing of L, and since the number of configurations is $m^W = O(n^{O(1/\varepsilon^d)})$, we get the running time from Lemma 1.2. The value of an optimal schedule can be guessed, within factor $1+\varepsilon$, by obtaining first a $(d+1)$-approximate solution. This can be done by applying an approximation algorithm for resource constrained scheduling due to [20].

□

3.2 LP Rounding Combined with Enumeration

As described in Section 2.1, a common technique for obtaining a PTAS is to extend all possible solutions for small subsets of elements. This technique can be combined with LP rounding as follows. Repeatedly select a small subset of input elements, $S_g \subseteq I$, to be the basis for an approximate solution; solve an LP for the remaining elements, $I \setminus S_g$. Select the subset S_g which gives the best solution. We exemplify the usage of the technique to obtain a PTAS for *covering integer programs with multiplicity constraints (CIP)*. In this core problem, we must fill up an R-dimensional bin by selecting (with bounded number of repetitions) from a set of n R-dimensional items, such that the overall cost is minimized. Formally, let $A = \{a_{ji}\}$ denote the sizes of the items in the R dimensions, $1 \leq j \leq R$, $1 \leq i \leq n$; the cost of item i is $c_i \geq 0$. Let x_i denote the number of copies selected from item i, $1 \leq i \leq n$. We seek an n-vector $\mathbf{x}$ of non-negative integers, which minimizes $c^T\mathbf{x}$, subject to the R constraints given by $A\mathbf{x} \geq b$, where $b_j \geq 0$ is the size of the bin in dimension

j. In addition, we have multiplicity constraints for the vector $\mathbf{x}$, given by $\mathbf{x} \leq d$, where $d \in \{1, 2, \ldots\}^n$.

Covering integer programs form a large subclass of integer programs encompassing such NP-hard problems as minimum knapsack and set cover. This implies the hardness of CIP in fixed dimension (i.e., where R is a fixed constant). For general instances, the hardness of approximation results for set cover carry over to CIP. Comprehensive surveys of known results for CIP and CIP_∞, where the multiplicity constraints are omitted, are given in [44] and in [35] (see also in [36]).

We describe below a PTAS for CIP in fixed dimension. The scheme presented in [44] builds on the classic LP-based scheme due to Frieze and Clarke for the R-dimensional knapsack problem [17]. Consider an instance of CIP in fixed dimension, R. We want to minimize $\sum_{i=1}^{n} c_i x_i$ subject to the constraints $\sum_{i=1}^{n} a_{ij} x_i \geq b_j$ for $j = 1, \ldots, R$, and $x_i \in \{0, 1, \ldots d_i\}$ for $i = 1 \ldots, n$.

Assume that we know the optimal cost, C, for the CIP instance. The scheme of [44] uses a reduction to the binary *minimum R-dimensional multiple choice knapsack (R-MMCK)* problem. For some $R \geq 1$, an instance of binary R-MMCK consists of a single R-dimensional knapsack, of size b_j in the j-th dimension, and m sets of items. Each item has an R-dimensional size and is associated with a cost. The goal is to pack a subset of items, by selecting at most one item from each set, such that the total size of the packed items in dimension j is at least b_j, $1 \leq j \leq R$, and the overall cost is minimized.

Given the value of C, the parameter ε and a CIP instance with bounded multiplicity, the scheme constructs an R-MMCK instance in which the knapsack capacities in the R dimensions are b_j, $1 \leq j \leq R$. Also, there are n sets of items denoted by A^i, $1 \leq i \leq n$. Let $\hat{K}^i$ be the integer value satisfying $d_i c_i \in [\hat{K}^i \varepsilon C/n, (\hat{K}^i + 1)\varepsilon C/n)$, then the number of items in A^i is $K^i = \min(\hat{K}^i, \lfloor n/\varepsilon \rfloor)$. The set A^i represents all possible values which x_i can take in the solution for CIP. In particular, the k-th item in A^i, denoted (i, k), represents the assignment of a value in $[0, d_i]$ to x_i, such that $c(i, k)$, the total cost incurred by item i is in $[k\varepsilon C/n, (k + 1)\varepsilon C/n)$. This total cost is rounded down to the nearest integral multiple of $\varepsilon C/n$; thus, $c(i, k) = k\varepsilon C/n$. The size of the item (i, k) in dimension j, $1 \leq j \leq R$, is given by $s_j(i, k) = a_{ij}$.

Given an instance of R-MMCK, guess a partial solution, given by a small size set, S; these items have the maximal costs in some optimal solution. The size of S is a fixed constant, namely, $|S| = h = \lfloor \frac{2R(1+\varepsilon)}{\varepsilon} \rfloor$. The set S will be extended to an *approximate* solution, by solving a linear program for the remaining items.

The value of h is chosen such that the resulting solution is guaranteed to be within $1+\varepsilon$ from the optimal, as computed below. Let $E(S)$ be the subset of items with costs that are larger than the minimal cost of any item in S, that is, $E(S) = \{(i,k) \notin S \mid c(i,k) > c_{min}(S)\}$, where $c_{min}(S) = \min_{(i,k)\in S} c(i,k)$. Select all the items $(i,k) \in S$, and eliminate from the instance all the items $(i,k) \in E(S)$ and the sets A^i from which an item has been selected. In the next step we find an optimal *basic solution* for the following linear program, in which $x_{i,k}$ is an indicator variable for the selection of the item (i,k).

$$
\begin{array}{lll}
(LP(S)) \qquad \text{minimize} & \displaystyle\sum_{i=1}^{n}\sum_{k=1}^{K^i} x_{i,k}\cdot c(i,k) & \\
\textit{subject to}: & \displaystyle\sum_{k=1}^{K^i} x_{i,k} \le 1 \quad \text{for } i=1,\ldots,n, & \\
& \displaystyle\sum_{i=1}^{n}\sum_{k=1}^{K^i} s_j(i,k)x_{i,k} \ge b_j \quad \text{for } j=1,\ldots,R & \\
& 0 \le x_{i,k} \le 1 \quad \text{for } (i,k) \notin S \cup E(S) & \\
& x_{i,k} = 1 \quad \text{for } (i,k) \in S & \\
& x_{i,k} = 0 \quad \text{for } (i,k) \in E(S) &
\end{array}
$$

Given an optimal fractional solution for the above program, we get an integral solution as follows. For any i, $1 \le i \le n$, let $k_{max} = k_{max}(i)$ be the maximal value of $1 \le k \le K^i$ such that $x_{i,k} > 0$, then we set $x_{i,k_{max}} = 1$ and, for any other item in A^i, $x_{i,k} = 0$. Finally, we return to the CIP instance and assign to x_i the maximum value for which the total (rounded down) cost for item i is $c(i,k_{max})$.

The next three lemmas show that the scheme yields a $(1+\varepsilon)$-approximation to the optimal cost, and that the resulting integral solution is feasible.

Lemma 1.4 *If there exists an optimal (integral) solution for CIP with cost C, then the integral solution obtained from the rounding for R-MMCK has the cost $\hat{z} \le (1+\varepsilon)C$.*

Proof

Let $\mathbf{x}^*$ be an optimal (fractional) solution for the linear program LP(S), and let S^* be the corresponding subset of items, that is, $S^* = \{(i,k) |\ x^*_{i,k} = 1\}$. If $|S^*| < h$ then we are done: in some iteration, the scheme will try S^*; otherwise, let $S^* = \{(i_1,k_1),\ldots,(i_g,k_g)\}$, such that $c(i_1,k_1) \ge \cdots \ge c(i_g,k_g)$, for some $g > h$. Let $S^*_h = \{(i_1,k_1),\ldots,(i_h,k_h)\}$, and $\sigma = \sum_{t=1}^{h} c(i_t,k_t)$. Then, for any item $(i,k) \notin (S^*_h \cup E(S^*_h))$, we have

$c(i,k) \leq \sigma/h$. Let $z^*, \hat{z}$ denote the optimal (integral) solution and the solution output by the scheme for the R-MMCK instance, respectively. Denote by $\mathbf{x}^B(S_h^*), \mathbf{x}^I(S_h^*)$ the basic and integral solutions of LP(S) as computed by the scheme, for the initial guess S_h^*.

By the above rounding method, for any $1 \leq i \leq n$, the cost of the item selected from A^i is $c(i, k_{max})$. Let F denote the set of items for which the basic variable was a fraction, that is, $F = \{(i,k) |\ x_{i,k}^B(S_h^*) < 1\}$, and let $\delta = \sum_{(i,k)\in F} c(i,k)$.

Then, we get that

$$z^* \geq \sum_{i=1}^{n}\sum_{k=1}^{K^i} c(i,k)x_{i,k}^B(S_h^*)$$

$$\geq \sum_{i=1}^{n}\sum_{k=1}^{K^i} c(i,k)x_{i,k}^I(S_h^*) - \delta.$$

Recall that in any *basic* solution for a linear program, the number of non-zero variables is bounded by the number of tight constraints in some optimal solution. Assume that in the optimal (fractional) solution of $LP(S_h^*)$ there are L tight constraints, where $0 \leq L \leq n + R$. Then in the basic solution $\mathbf{x}^B(S_h^*)$, at most L variables can be strictly positive. Thus, at least $L - 2R$ variables get an integral value (i.e. '1'), and $|F| \leq 2R$. Note that for any $(i,k) \in F$, $c(i,k) \leq \sigma/h$, since $F \cap (S_h^* \cup E(S_h^*)) = \emptyset$. Hence, we get that $z^* \geq \hat{z} + \frac{2R\sigma}{h} \geq \hat{z} + \frac{2R\hat{z}}{h} \geq \frac{\hat{z}}{1+\varepsilon}$. □

The next two lemmas follow from the rounding method used by the scheme.

Lemma 1.5 *The scheme yields a feasible solution for the CIP instance.*

Lemma 1.6 *The cost of the integral solution for the CIP instance is at most $\hat{z} + \varepsilon C$.*

Note that C can be guessed in polynomial time within factor $(1+\varepsilon)$, using binary search over the range $(0, \sum_{i=1}^{n} d_i c_i)$. Thus, combining the above lemmas we get:

Theorem 1.7 *There is a polynomial time approximation scheme for CIP in fixed dimension.*

Consider now the special case where the multiplicity constraints are omitted; that is, each variable x_i can get any non-negative (integral) value. For this special case, we can use a linear programming formulation in which the number of constraints is R, which is fixed. A PTAS for this problem can be derived from the

scheme of Chandra et al. [10] for integer multidimensional knapsack. Drawing from recent results for CIPs, we describe below the PTAS of [44], which improves the running time in [10] by using a fast approximation scheme for solving the linear program.

A Scheme for CIP_∞ The scheme, called below *multi-dimensional cover with parameter ε (MDC_ε)*, proceeds in the following steps.

(i) For a given $\varepsilon \in (0,1)$, let $\delta = \lceil R \cdot ((1/\varepsilon) - 1) \rceil$.

(ii) Renumber the items by $1, \ldots, n$, such that $c_1 \geq c_2 \geq \cdots \geq c_n$.

(iii) Denote by Ω the set of integer vectors $\mathbf{x} = (x_1, \ldots, x_n)$ satisfying $x_i \geq 0$, and $\sum_{i=1}^{n} x_i \leq \delta$. For any vector $\mathbf{x} \in \Omega$: Let $d \geq 1$ be the maximal integer i for which $x_i \neq 0$. Find a $(1+\varepsilon)$-approximation to the optimal (fractional) solution of the following linear program.

$$
\begin{aligned}
(LP') \qquad & \text{minimize} && \sum_{i=d+1}^{n} c_i z_i \\
& \textit{subject to :} && \sum_{i=d+1}^{n} a_{ij} z_i \geq b_j - \sum_{i=1}^{n} a_{ij} x_i \quad \text{for } j = 1, \ldots, R \\
& && z_i \geq 0, \quad \text{for } i = d+1, \ldots, n
\end{aligned}
\tag{1.5}
$$

The constraints (1.5) reflect the fact that we need to fill in each dimension j at least the capacity $b_j - \sum_{i=1}^{n} a_{ij} x_i$, once we obtained the vector $\mathbf{x}$.

Let $\hat{z}_i$, $d+1 \leq i \leq n$, be a $(1+\varepsilon)$ -approximate solution for LP'. We take $\lceil \hat{z}_i \rceil$ as the integral solution. Denote by $C_{MDC}(\mathbf{x}) = \sum_{i=d+1}^{n} c_i \lceil \hat{z}_i \rceil$ the value obtained from the rounded solution, and let $c(\mathbf{x}) = \sum_{i=1}^{n} c_i x_i$.

(iv) Select the vector $\mathbf{x}^*$ for which $C_{MDC_\varepsilon}(\mathbf{x}^*) = \min_{\mathbf{x}}(c(\mathbf{x}) + C_{MDC}(\mathbf{x}))$.

We now show that MDC_ε is a PTAS for CIP_∞. Let C_o be the cost of an optimal integral solution for the CIP_∞ instance.

Theorem 1.8 *MDC_ε is a PTAS for CIP_∞ which satisfies the following. (i) If $C_o \neq 0, \infty$ then $C_{MDC_\varepsilon}/C_o < 1+\varepsilon$. (ii) The running time of algorithm MDC_ε is $O(n^{\lceil R/\varepsilon \rceil} \cdot \frac{1}{\varepsilon^2} \log C)$, where $C = \max_{1 \leq i \leq n} c_i$ is the maximal cost of any item, and its space complexity is $O(n)$.*

The next lemma is required to prove the theorem.

Lemma 1.7 *For any $\varepsilon > 0$, a $(1+\varepsilon)$-approximation to the optimal solution for LP' can be found in $O(1/\varepsilon^2 R \log(C \cdot R))$ steps.*

Proof

For a system of inequalities as given in LP', there is a solution in which at most R variables get non-zero values. This follows from the fact that the number of non-trivial constraints is R. Hence, it suffices to solve LP' for the $\binom{n-d}{R}$ possible subsets of R variables, out of $(z_{d+1}, \ldots, z_n)$. This can be done in polynomial time since R is fixed. Now, for each subset of R variables, we have an instance of the *fractional covering problem*, for which we can use a fast approximation scheme (see, e.g., in [16]) to obtain a $(1+\varepsilon)$-approximate solution.

□

Proof of Theorem 1.8: For showing (i), assume that the optimal (integral) solution for the CIP_∞ instance is obtained by the vector $\mathbf{y} = (y_1, \ldots, y_n)$. If $\sum_{i=1}^{n} y_i \leq \delta$ then $C_{MDC_\varepsilon} = C_o$, since in this case $\mathbf{y}$ is a valid solution, and $\mathbf{y} \in \Omega$, therefore, in some iteration MDC_ε will examine $\mathbf{y}$. Suppose that $\sum_{i=1}^{n} y_i > \delta$, then we define the vector $\mathbf{x} = (y_1, \ldots, y_{d-1}, x_d, 0, \ldots, 0)$, such that $y_1 + \cdots + y_{d-1} + x_d = \delta$. (Note that $x_d \neq 0$.) Let $\tilde{C}_o(\mathbf{x}) = \sum_{i=d+1}^{n} c_i \hat{z}_i$ be the approximate fractional solution for LP'. We have that $\mathbf{x} \in \Omega$, therefore

$$C_{MDC}(\mathbf{x}) - \tilde{C}_o(\mathbf{x}) \leq R c_d, \tag{1.6}$$

Let $C_o(\mathbf{x})$ be the optimal fractional solution for LP' with the vector $\mathbf{x}$. Note that C_o, the optimal (integral) solution for CIP_∞, satisfies

$$C_o > c(\mathbf{x}) + C_o(\mathbf{x}), \tag{1.7}$$

since $C_o(\mathbf{x})$ is a lower bound for the cost incurred by the integral values $y_{d+1}, \ldots, y_n$. In addition,

$$c(\mathbf{x}) + C_{MDC}(\mathbf{x}) \geq C_{MDC_\varepsilon}. \tag{1.8}$$

Hence, we get that

$$\begin{aligned}\frac{C_o}{C_{MDC_\varepsilon}} &\geq \frac{c(\mathbf{x})+C_o(\mathbf{x})}{c(\mathbf{x})+C_{MDC}(\mathbf{x})} > 1-\frac{C_{MDC}(\mathbf{x})-C_o(\mathbf{x})}{c(\mathbf{x})+C_{MDC}(\mathbf{x})-C_o(\mathbf{x})} \\ &\geq 1-\frac{C_{MDC}(\mathbf{x})-\tilde{C}_o(\mathbf{x})(1-\varepsilon)}{c(\mathbf{x})+C_{MDC}(\mathbf{x})-\tilde{C}_o(\mathbf{x})} \\ &\geq (1-\varepsilon)\left(1-\frac{C_{MDC}(\mathbf{x})-\tilde{C}_o(\mathbf{x})}{c(\mathbf{x})+C_{MDC}(\mathbf{x})-\tilde{C}_o(\mathbf{x})}\right) \\ &\geq (1-\varepsilon)(1-\frac{C_{MDC}(\mathbf{x})-\tilde{C}_o(\mathbf{x})}{\delta c_d+C_{MDC}(\mathbf{x})-\tilde{C}_o(\mathbf{x})})\end{aligned}$$

The first inequality follows from (1.7) and (1.8), and the third inequality follows from the fact that $\tilde{C}_o(\mathbf{x})(1-\varepsilon) \leq C_o(\mathbf{x}) \leq \tilde{C}_o(\mathbf{x})$. The last inequality follows from the fact that $c(\mathbf{x}) \geq \delta c_d$.

Using (1.6), we get that $\frac{C_o}{C_{MDC_\varepsilon}} \geq (1-\varepsilon)1 - Rc_d/(\delta c_d + Rc_d) \geq (1-\varepsilon)^2$. Taking in the scheme $\tilde{\varepsilon} = \varepsilon/2$, we get the statement in (i).

Next, we show (ii). Note that $|\Omega| = O(n^\delta)$ since the number of possible choices of n non-negative integers, whose sum is at most δ is bounded by $\binom{n+\delta}{\delta}$. Now, given a vector $\mathbf{x} \in \Omega$, we can compute $C_{MDC}(\mathbf{x})$ in $O(n^R)$ steps since at most R variables out of $z_{d+1}, \ldots, z_n$ can have non-zero values. Multiplying by the complexity of the FPTAS for fractional covering, as given in Lemma 1.7, we get the statement of the theorem. □

Enumeration is combined with LP rounding also in the PTAS of Caprara et al. [9] for the *knapsack problem with cardinalities constraints*, and in a PTAS for the *multiple knapsack* problem due to Chekuri and Khanna [12], among others. The scheme of [9] is based on the scheme of Frieze and Clarke [17], with the running time improved by factor of n, the number of items. The scheme of [17] is the basis also for PTASs for other variants of the knapsack problem. (A comprehensive survey is given in [31]; see also in [45].)

4 Approximation Schemes for Geometric Problems

In this Section we present approximation techniques that are specialized for geometric optimization problems. For a complete description of these techniques we refer the reader to the survey by Arora [3], Chapter 11 in [47], and Chapters 8 and 9.3.3 in [24]. A typical input for a geometric problem is a set of elements in the

space (such as points in the plane); the goal is to connect or pack these elements in a way that minimizes the resources used (e.g., total length of connecting lines, total number of covering objects).

4.1 Randomized Dissection

We present below the techniques used in the PTAS of Arora [2] for the Euclidean Traveling Salesman Problem (TSP). In the classical TSP problem, given are non-negative edge weights for the complete graph K_n, and the goal is to find a tour of minimum cost, where a tour refers to a cycle of length n. In other words, the goal is to find an ordering of the nodes such that the total cost of the edges along the path visiting all nodes according to this ordering is minimal. In general, TSP is NP-hard in the strong sense, and it cannot be approximated within any multiplicative factor $c > 1$, unless P = NP. The PTAS of Arora considers the relaxed problem of *Euclidean* TSP. The input is a set of n points in $\Re^d$, and the edge weights are the Euclidean (ℓ_2) distances between them.

The idea of the PTAS is to dissect the plane into squares, and to look (using dynamic programming) for a tour that crosses the resulting grid lines only at specific points, denoted *portals*. The parameter ε of the PTAS determines the depth of the recursive dissection, as well as the density of the portals. A smaller ε results in more portals and a finer dissection, which lead to a less restricted tour and a larger dynamic programming instance. Randomization is used to determine an initial shift of the grid lines.

A dissection of a square is a recursive partitioning into squares. It can be viewed as a tree of squares whose root is the square we started with. Each square in the tree is partitioned into four equal squares, which are its children. The leaves are squares of a small sidelength - determined by the parameter ε of the PTAS.

The location of the grid lines is determined *randomly* as follows. Given a set of n points in $\Re^2$, enclose the points in a minimum bounding square. Let ℓ be the side of this square. Let $p \in \Re^2$ be the lower left endpoint of the bounding box. Enclose the bounding box inside a larger square, denoted the *enclosing box* of sidelength $L = 2\ell$, and position the enclosing box such that p has distance a from the left edge and b from the lower edge, where $a, b \leq \ell$ are chosen randomly. The randomized dissection is the dissection of this enclosing box. Note that the randomness is used only to determine the placement of the enclosing box (and

its accompanying dissection).

We now describe the PTAS in [2] for the Euclidean TSP problem, which uses the above randomized dissection. Formally, for every $\varepsilon > 0$, this PTAS finds a $(1+\varepsilon)$-approximation to Euclidean TSP.

First, perform randomized dissection to the bounding box of the n points. Recall that L is the side of the enclosing box. The recursive procedure of subdividing the squares stops when the side lengths of the squares becomes less than $L\varepsilon/8n$, or when each square at the last level contains at most one point. We may assume (by scaling) that L is a power of 2 and that the sides of squares at the last level are unit length. Thus, at most $\log L$ iterations are required, and $L \leq 8n/\varepsilon$. When there is more than one point in a unit square, consolidate them into one new "bigger" point. Any tour for the resulting set of points can be augmented to a tour for the original set of points with an increase in length bounded by $\sqrt{2}nL\varepsilon/8n$, which is negligible, since $L \leq OPT/2$. Henceforth, we shall assume that there is at most one point per unit square.

The *level* of a square in the dissection is its depth in the recursive dissection tree; the root square has level 0. We also assign a level from 0 to $\log(L-1)$ to each horizontal and vertical grid line that participates in the dissection. The horizontal (resp., vertical) line that divides the enclosing box into two has level 0. Similarly, the 2^i horizontal and 2^i vertical lines that divide the level i squares into level $i+1$ squares have level i. The following property of a randomized dissection is used: Any fixed vertical grid line that intersects the bounding box of the instance has probability $\frac{2^i}{\ell} = \frac{2^{i+1}}{L}$ to be a line at level i.

Next, the location of the portals is determined. Let $m = \frac{1}{\varepsilon}\log L$. The parameter m is the *portal parameter* which determines the density of the points the path can pass through. A level i line has $2^{i+1}m$ equally spaced portals. In addition, we also refer to the corners of each square as a portal. Since a level i line has 2^{i+1} level $i+1$ squares touching it, it follows that each side of the square has at most $m+2$ portals (m regular portals plus the 2 corners), and a total of at most $4m+4$ portals on its boundary. A *portal-respecting tour* is one that, whenever it crosses a grid line, does so at a portal.

Finally, dynamic programming is used to find the optimum portal-respecting tour in time $2^{O(m)}L\log L$. Since $m = O(logn/\varepsilon)$, we get a total running time of $n^{O(1/\varepsilon)}$. The dynamic programming as well as the complete analysis of bounding the PTAS error and the time complexity are given in [3].

Note that since the PTAS uses randomization, the error of the PTAS is a random variable. Formally,

let OPT denote the cost of the optimum salesman tour and $OPT_{a,b,m}$ denote the cost of the best portal-respecting tour when the portal parameter is m and the random shifts are a, b, then,

Theorem 1.9 *The expectation (over the choices of a, b) of $OPT_{a,b,m} - OPT$ is at most $2logL/mOPT$, where L is the sidelength of the enclosing box.*

As mentioned in the survey of Arora [3], this method of dissection can be used to develop PTASs for other geometric optimization problems such as minimum Steiner tree, facility location with capacities and demands, and Euclidean min-cost k-connected subgraph.

Another class of geometric optimization problem is the class of *clustering* problems, such as metric max-cut and k-median. In recent research on clustering problems, a core idea in the design of approximation schemes is to use random sampling of data points from a biased distribution, which depends on the pairwise distances. This technique is used, e.g., in the PTAS of Fernandez de la Vega and Kenyon for metric max-cut [19], and in the work of Indyk on metric 2-clustering [26]. For more details on the technique and its applications, we refer the reader to [18].

4.2 Shifted Plane Partitions

The *shifting* technique that is applied to geometric problems is based on selecting the best solution over a (polynomial size) set of feasible solutions. Each candidate feasible solution is obtained using a divide-and-conquer approach, in which the plane is partitioned into disjoint areas (strips). The technique can be applied to geometric problems such as square packing or covering with disks, which arise in VLSI design, image processing and many other important areas. A common goal in these problems is to cover or pack elements (e.g., points in the plane) into a minimal number of objects (e.g., squares of given size).

Recall that each candidate solution is obtained by using divide-and-conquer approach, in which the plane is partitioned into strips. A solution for the original problem is formed by taking the union of the solutions for these strips. Consecutive solutions refer to consecutive partitions of the plane into strips, which differ from each other by *shifting the partitioning bars*, using the shifting parameter. The smaller the shifting parameter, the larger is the number of candidate solutions to be considered, and the better resulting approximation.

We illustrate the shifting technique for the problem of covering n points in the 2-dimensional plane. The

complete analysis is given in [25, 24]. Assume that the n points are enclosed in an area I. The goal is to cover these points with a minimal number of disks of diameter D. Denote by ℓ the shifting parameter. The area I is divided into vertical strips of width D. Each set of ℓ consecutive strips are grouped together to form strips of width ℓD. Note that there are ℓ different ways to determine this grouping - and they can derive from each other by shifting the partitioning bars to the right over distance D. Denote the ℓ distinct partitions obtained this way by $S_1, S_2, ..., S_\ell$.

Let A be an algorithm to solve the covering problem on strips of width at most ℓD. The algorithm A can be used to generate a solution for a given partition S_j. We apply A to each strip in S_j and then union the sets of disks used. The *shift algorithm*, s_A, defined for a given A, uses A to solve the problem for the ℓ possible partitions and selects the solution that requires minimum number of disks.

The following lemma gives the performance ratio of s_A (denoted r_{s_A}) as function of ℓ and the performance ration of A (denoted r_A).

Lemma 1.8

$$r_{s_A} \leq r_A(1 + \frac{1}{\ell}).$$

The algorithm A may itself be derived from an application of the shifting technique. In our example, in order to solve the covering problem on a strip of width ℓD, the strip is cut into squares of size $\ell D \times \ell D$, for which an optimal solution can be found by exhaustive search.

We note that the above shifting technique can be used to derive PTASs for several other problems, including minimum vertex-cover and maximum independent-set in planar graphs [6]. The idea is that a planar graph can be decomposed into components of bounded outer-planarity. The solution for each component can be found using dynamic programming. The shifting idea is to remove one 'layer' from the graph in each iteration. This removal guarantees that the number of cross-cluster edges is small, so by considering the union of the local cluster solutions one can get a good approximation for the original problem.

5 Concluding Remarks

There are many other interesting applications of the techniques described in this chapter. We mention a few of them. Golubchik et al. apply enumeration to a *structured* instance in solving the problem of data placement on disks (see also in [30]). The technique of extending solutions for small subsets is applied by Khuller at al. [34] to the problem of broadcasting in heterogeneous networks. Kenyon et al. [32] used a non-trivial combination of grouping with periodic scheduling to obtain a PTAS for data broadcast.

As mentioned in Section 4, some techniques are specialized for certain types of problems. For graph problems, some PTASs exploit the density of the input graph (see, e.g., [5]). There are PTASs which build on the properties of *planar graphs* (see, e.g., [22, 14]).

Finally, we have mentioned in Sections 2.3 and 4 some techniques used in *randomized* approximation schemes. A detailed exposition of randomized approximation schemes for counting problems is given in Chapter 11 in [38] (see also Chapter R-8 in this book). Benczúr and Karger presented in [7] randomized approximation schemes for cuts and flows in capacitated graphs. Efraimidis and Spirakis used in [15] the technique of *filtered randomized rounding* in developing randomized approximation schemes for scheduling unrelated parallel machines.

References

[1] F.N. Afrati, E. Bampis, C. Chekuri, D.R. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenko, Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates. In *Proc of FOCS*, 32–44, 1999.

[2] S. Arora Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM, 45(5):753–782, 1998.

[3] S. Arora, Approximation schemes for NP-hard Geometric Optimization Problems: A survey. *Math Programming*, 2003.

[4] S. Arora and G. Karakostas, Approximation Schemes for Minimum Latency Problems. *SIAM J. Comput.*, 32(5), 1317–1337, 2003.

[5] S. Arora, D, Karger and M. Karpinski. Polynomial Time Approximation Schemes for Dense Instances of NP-Hard Problems. In *Proc. of STOC*, 1995.

[6] B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs, J. of ACM, vol. 41(1), pp. 153–180, 1994.

[7] A.A. Benczúr and D.R. Karger. Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs. Technical Report, MIT, July 2002.

[8] N. Bansal and M. Sviridenko. Two-dimensional bin packing with one dimensional resource augmentation. Submitted for publication.

[9] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333– 345, 2000.

[10] A.K. Chandra, D.S. Hirschberg and C.K. Wong, "Approximate Algorithms for Some Generalized Knapsack Problems". *Theoretical Computer Science* 3, pp. 293–304, 1976.

[11] C. Chekuri and S. Khanna. On Multidimensional Packing Problems. *SIAM J. Comput.* 33(4): 837–851, 2004.

[12] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *Proc. of SODA*, 213–222, 2000.

[13] C. Chekuri and S. Khanna. Approximation Schemes for Preemptive Weighted Flow Time. In *Proc. of STOC*, 297–305, 2002.

[14] E.D. Demaine and M. Hajiaghayi, "Bidimensionality: New Connections between FPT Algorithms and PTASs," In *Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 590–601, 2005.

[15] P. Efraimidis and P.G. Spirakis. Randomized Approximation Schemes for Scheduling Unrelated Parallel Machines. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(7), 2000.

[16] L. Fleischer, "A Fast Approximation Scheme for Fractional Covering Problems with Variable Upper Bounds". In *Proc. of SODA*, pp. 994–1003, 2004.

[17] A. M. Frieze and M.R.B. Clarke, Approximation Algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. In *European J. of Operational Research*, 15(1):100–109, 1984.

[18] W. Fernandez de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani, Approximation Schemes for Clustering Problems. In Proc. of *STOC*, 2003.

[19] W. Fernandez de la Vega and C. Kenyon. A randomized approximation scheme for metric MAX-CUT. Proc. 39th IEEE Symp. on Foundations of Computer Science, pp 468–471, 1998.

[20] M.R. Garey and R.L. Graham Bounds for Multiprocessor Scheduling with Resource Constraints. SIAM J. Comput. 4(2): 187–200, 1975.

[21] R.L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[22] M. M. Halldórsson and G. Kortsarz. Tools for Multicoloring with Applications to Planar Graphs and Partial k-Trees. *Journal of Algorithms* 42(2), 334–366, 2002.

[23] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of the ACM*, 34(1):144–162, 1987.

[24] D.S. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PUS Publishing Company, 1995.

[25] D.S. Hochbaum and W. Maass. *Approximation schemes for covering and packing problems in image processing and VLSI Journal of the ACM*, 32(1):130 – 13, 1985.

[26] P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. In Proc. FOCS '99

[27] K. Jansen, R. Solis-Oba, M. Sviridenko, Makespan Minimization in Job Shops: A Linear Time Approximation Scheme. *SIAM J. Discrete Math.* 16(2), 288–300, 2003.

[28] K. Jansen, M. Sviridenko, Polynomial Time Approximation Schemes for the Multiprocessor Open and Flow Shop Scheduling Problem. In Proc. of STACS, 455–465, 2000.

[29] D. Karger, C. Stein and J. Wein. Scheduling Algorithms, In *CRC Handbook on Algorithms*, 1997.

[30] S. Kashyap and S. Khuller, Algorithms for Non-Uniform Size Data Placement on Parallel Disks. *FST & TCS*, 2003.

[31] H. Kellerer, U. Pferschy and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[32] C. Kenyon, N. Schabanel, N.E. Young, Polynomial-Time Approximation Scheme for Data Broadcast. *CoRR cs.DS/0205012*, 2002.

[33] S. Khanna and R. Motwani, Towards a Syntactic Characterization of PTAS, *Proc. of STOC*, 329–337, 1996.

[34] S. Khuller, Y. Kim and G. Woeginger, A polynomial time approximation scheme for broadcasting in heterogeneous networks. *Proc. of APPROX*, 2004.

[35] S. G. Kolliopoulos, "Approximating covering integer programs with multiplicity constraints", *Discrete Applied Math.*, 129:2–3, 461 –473, 2003.

[36] S. G. Kolliopoulos and N. E. Young, "Tight Approximation Results for General Covering Integer Programs". In *Proc. of FOCS*, 522–528, 2001.

[37] R. Motwani. Lecture notes on approximation algorithms. Technical report, Dept. of Computer Science, Stanford Univ., CA, 1992.

[38] R. Motwani and P. Raghavan. *Randomized Algorithms* Cambridge Univ. Press, 1995.

[39] C.N. Potts. Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics* 10, pp.155–164, 1985.

[40] S. Sahni. Approximate Algorithms for the 0/1 knapsack problem, *J. of the ACM*, 22, 115–124, 1975.

[41] S. Sahni. Algorithms for scheduling independent tasks, *J. of the ACM*, 23, 555–565, 1976.

[42] P. Schuurman and G.J. Woeginger. Approximation Schemes - A Tutorial. To appear in the book "Lectures on Scheduling", edited by R.H. Möehring, C.N. Potts, A.S. Schulz and G.J. Woeginger, LA Wolsey.

[43] S.V. Sevastianov and G.J. Woeginger. Makespan minimization in open shops: A polynomial time approximation scheme. *Mathematical Programming*, Vol. 82, 191–198, 1998.

[44] H. Shachnai, O. Shmueli and R. Sayegh. Approximation Schemes for Deal Splitting and Covering Integer Programs with Multiplicity Constraints, *Proc. of WAOA*, 111–125, 2004.

[45] H. Shachnai and T. Tamir, "Approximation Schemes for Generalized 2-dimensional Vector Packing with Application to Data Placement". In *Proc of APPROX*, August 2003.

[46] H. Shachnai and T. Tamir and O. Yehezkely, Approximation Schemes for Packing with Item Fragmentation. *3rd Workshop on Approximation and Online Algorithms (WAOA)*, October 2005.

[47] V. Vazirani, Approximation algorithms, Springer Verlag, 2001.

[48] G.J. Woeginger, "There is no Asymptotic PTAS for two-dimensional vector packing", *Information Processing Letters*, 64, pp. 293–297, 1997.