Critical examination of important combinatorial techniques and investigation of their combination with methods from continuous optimization
Start date of the project: January 2004
Duration: 48 months
Project Coordinator: Prof. Dr. math. Friedhelm Meyer auf der Heide
Heinz Nixdorf Institute, University of Paderborn, Germany

Due date of deliverable: December 2004
Actual submission date: January 2005
Re-submission date: April 2005
Version: 2
Dissemination level: PU – public

Work Package 3.1: Approximation Theory for Large-Scale Optimization

Participants: Max Planck Institute for Computer Science (MPII), Saarbrücken, Germany
University of Paderborn (UPB), Germany
Computer Technology Institute (CTI), Patras, Greece
Charles University (CUNI), Prague, Czech Republic
Rheinisch-Westfälische Technische Hochschule (RWTH), Aachen, Germany
University of Cyprus (UCY), Cyprus

Authors of deliverable: Holger Bast (bast@mpi-sb.mpg.de)
René Beier (rbeier@mpi-sb.mpg.de)
Martin Gairing (gairing@upb.de)
Petr Kolman (kolman@kam.mff.cuni.cz)
Ulrich Meyer (umeyer@mpi-sb.mpg.de)
Joachim Reichel (reichel@mpi-sb.mpg.de)
Paul Spirakis (spirakis@cti.gr)
George Tsaggouris (tsaggour@ceid.upartras.gr)
1 Introduction

Combinatorial optimization refers to that subfield of optimization where the set of feasible solutions is a finite set that can be easily enumerated. Unlike optimization problems in general, combinatorial optimization problems are therefore always, at least in principle, computable, by performing a simple so-called exhaustive search: go through the set of feasible solutions, compute the objective function for each of them, and pick one at which the optimum value is reached. In practice, however, the set of feasible solutions is very large already for relatively small input sizes. More precisely, there are typically at least $\Omega(n!)$ feasible solutions for inputs of size $n$. This prohibits an exhaustive search.

Research on efficient combinatorial optimization has a long tradition and a very strong theoretical foundation. The purpose of this deliverable is to examine some of the most important existing approaches with respect to their strengths and weaknesses, with respect to possible extensions to the non-linear techniques and problems we discussed in Deliverable 3.1.1, and with respect to their applicability and usefulness for the problems encountered within DELIS. The deliverable is organized into three parts: linear programming, network flows, and non-standard objective functions.

1.1 Linear programming

In Section 2, we discuss a number of techniques that extend or complement the rich theory on linear programming. We briefly recall that in a linear program the set of feasible solutions is given by set of linear inequalities, and the objective function is linear as well.

In Section 2.1, we consider linear programming scenarios with a very large number of inequalities; too large even to merely store them. We review various approaches from the literature to deal with this problem. One particular such approach, the method of Lagrangian relaxation, was successfully applied in recent work within DELIS.

In Section 2.2, we consider linear programs with an additional integrality constraint on the variables, so called integer linear programs (ILPs). We briefly recall the fundamental LP-rounding technique for reducing an ILP to an ordinary linear program, and sketch some novel developments that we have achieved within DELIS. We also discuss the issue of derandomizing the algorithms that are obtained by LP-rounding techniques.

In Section 2.3, we compare the effectiveness of the LP-rounding approach to that of an application-specific local search heuristic. We give an overview of the heuristic for a particular NP-complete network problem. While LP-rounding turned out to be unable to match the tight requirements imposed by the specific application, the local search gave satisfactory results.

1.2 Network flows

The second part of this deliverable is considered with network flows, where, roughly speaking, the task is to route as many units of flow as possible from given sources to given targets,
with the constraint that edges of the networks have certain bounded capacities. Network flow problems arise as explicit subproblems in many applications within DELIS. More generally, many fundamental combinatorial optimization problems actually have a natural formulation as a network flow problem.

Section 3.1 recalls two classical concepts of flow theory: blocking flows and the preflow-push idea from the classic works of Dinic [14] and Goldberg and Tarjan [22]. From these basic, yet elegant techniques a strong new result is established that at the same time simplifies and generalizes a whole thread of previous work.

In Section 3.2, we present a major advance in flow theory. The classical notion of a unit flow along a path is extended to the notion of a multflow along $k$ edge-disjoint paths. The fundamental theorem on the duality of flows and cuts is then extended to the generalized setting. It is sketched how these insights lead to new, fault-tolerant algorithms for various classical flow problems.

1.3 Non-standard objective functions

The last part of this deliverable deals with two non-standard objective functions that explicitly address two peculiarities of the type of problems we encounter in DELIS: that the data is so large that only a fraction of it fits into the main memory of a single computer, and that we know, e.g. from measurements (SP1), statistical properties of our input, which we would like to take advantage of for more efficient and more effective algorithms.

Section 4.1 is concerned with the issue of I/O-efficiency, where different costs and access types are assumed for data in caches, in main memory, on external disks, etc. We will briefly review the basics of this line of research, and then give the main idea of two new results on the fundamental network problems of shortest path computation and systematic graph traversal.

Section 4.2 briefly discusses the shortcomings of the traditional worst-case analysis, and then describes a new and very general framework that allows for more satisfactory analyses of a wide spectrum of combinatorial optimization problems, including such with a non-linear set of feasible solutions.

1.4 Further dissemination

This deliverable refers to a number of works that have been published [2] [12] [8] [9] [17] [19] or submitted for publications [20] [4] [6] [16] as part of the activities within DELIS in 2004. These can be obtained upon request from the coordinating site UPB.
2 Linear Programming

2.1 LPs with exponentially many variables

In many large-scale optimization problems, linear programming formulations include an exponential number of variables. Examples are network optimization problems, where for each network path one or more variables are needed in order to capture specific properties of it. Therefore such problems can not be efficiently solved by conventional LP solving methods (like simplex), because of the exponentially many variables. Note that even writing down the LP requires exponential time.

There are certain cases though where we can solve such problems efficiently by facilitating more sophisticated techniques. By the equivalence of optimization to separation, it is known that many problems of that sort can be efficiently solved to optimality via the ellipsoid method when we are able to identify the most violated constraint in the dual problem using a polynomial time oracle.

Another approach for such problems is column generation [13] and cutting plane techniques [25]. In the column generation technique instead of the original problem, a smaller instance is solved in each iteration. That instance is usually called the Restricted Master Problem (RMP) and considers only a small subset of the original problem variables. When the RPM is solved, the dual solution is employed to refine the representation. This is done by identifying one or more violated constraints (preferably the most violated one) in the dual of the original problem, and by adding the corresponding variables to the RMP. A possible direction to improve the running time of such an approach is to approximately solve the dual of the RMP. Experimental works show that this approach gives good running-time results in practice with little loss in optimality (see e.g., [23]).

Other approaches include identifying special characteristics in the problem and exploit these characteristics. An example is problems with non-negative coefficients, like packing and covering problems. For such problems a series of Lagrangian-relaxation based algorithms have been recently developed. Lagrangian-relaxation algorithms are based on the following idea: given an optimization problem specified as a collection of constraints, replace some of the constraints by some continuous “penalty” function. This function reflects the amount by which the constraint to be removed is violated: the more the constraint is violated the larger the value of the penalty function. These algorithms construct the solution iteratively in small steps by finding improvement directions, maintain the remaining constraints, while minimizing the increase in the penalty function. An example of such an algorithm is the algorithm for pure packing problems of Garg and Könemann [21]. A pure packing LP is a linear program of the form $\max\{c^T x | Ax \leq b, x \geq 0\}$, where $A_{m \times n}$, $b_{m \times 1}$ and $c_{n \times 1}$ have positive entries. The dual of that problem is $\min\{b^T y | A^T y \geq c, y \geq 0\}$. In that algorithm the exponential penalty function idea is employed. The dual variables are set exponential to the amount of violation of the corresponding primal constraint and in each iteration the “most violated” dual constraint is identified by a polynomial time oracle, and the corresponding primal variable is increased. In particular the Garg-Könemann approach is as follows.
The algorithm starts by setting \( x_0(j) = 0 \), \( \forall j = 1 \ldots n \), and \( y_0(i) = \frac{\delta}{b(i)} \), \( \forall i = 1 \ldots m \), for some small constant \( \delta \). In each iteration \( k \geq 1 \), the most violated constraint \( q = \arg\min_j \sum_i \frac{A(i,j)}{a(i,j)} y(i) \) in the dual is determined and the corresponding primal variable is increased by the quantity \( \Delta = \min_i \frac{b(i)}{A(i,q)} \), i.e., \( x_k(q) = x_{k-1}(q) + \Delta \), while the dual variables become \( y_k(i) = y_{k-1}(i) \left( 1 + \frac{\Delta}{b(i)/A(i,q)} \right) \), \( \forall i = 1 \ldots m \), for some small constant \( \varepsilon \). The algorithm terminates when \( b^T y > 1 \). By appropriately choosing \( \delta \), the algorithm computes an \((1 - \varepsilon)^2\) approximation in time \( O\left( \frac{1}{\varepsilon} (m \log_{1+\varepsilon} m) T_{orc} \right) \), where \( T_{orc} \) is the time required to compute the most violated constraint in the dual. Garg and Könemann apply these ideas to the maximum multicommodity flow problem. Even though the pure packing LP formulation that they use for this problem has an exponential number of variables, the most violated constraint in each iteration can be found by performing one shortest path computation per commodity, which yields a polynomial time oracle.

Fleischer [18] further improves on this technique by performing the algorithm steps in phases. Young later on [35] applies these ideas along with the idea of incrementing multiple variables simultaneously by Luby and Nisan [28], to give sequential and parallel approximation schemes for mixed packing and covering problems. Another recent work based on Lagrangian potential function methods is that of Bienstock and Iyengar in their paper for solving fractional packing problems in [11]. For a survey on potential function Lagrangian methods we refer the reader to [10]. The main advantage of such algorithms is that they result in fully polynomial time approximation schemes, and can be faster in practice and easier to implement in comparison to simplex, interior point or ellipsoid methods.

In some of the above mentioned methods the existence of an oracle that identifies a (or the most) violated constraint of the dual is assumed. But what can we do when a polynomial oracle does not exist for the problem under consideration? This is the case, for example, when the underlying subproblem of identifying the most violating constraint is NP-hard. In many cases however, NP-hard problems admit efficient approximation algorithms. Facilitating such algorithms is a promising direction in approximately solving such optimization problems.

### 2.2 LP-rounding and derandomization

Many optimization problems have a more or less natural modelling as a so-called integer linear program (ILP), that is, a set of linear inequalities and a linear objective function, and the additional constraint that all or at least some of the variables be integral. More specifically, for the following we assume that variables may merely take on the values 0 or 1; this is still sufficient for the vast majority of applications. While linear programs are known to have algorithms with a polynomial running time, the same is not true for integer linear programs, unless \( P = NP \).

One of the most frequently used approaches to solving integer linear programs is LP-rounding. It goes back to the idea of randomized rounding first published by Raghavan and Thompson [32], and became popular by the book of Motwani and Raghavan [29]. The basic idea is to temporarily drop the integrality constraint from an integer linear program, and solve the resulting ordinary linear program. This will in general result in some fractional solution, which
is then rounded to an integral solution. The challenge is to perform the rounding in such a way that the resulting integral solutions is feasible as well as close to what would have been an optimal solution for the integer linear program.

A strikingly simple, yet often effective technique here is randomized rounding. Assuming that all variables take on values between 0 and 1, each fractional value $x_i$ is rounded to 1 with probability $x_i$, and rounded to 0 with probability $1 - x_i$. This gives an integral solution, from which a feasible solution is then obtained by some local search heuristic.

In [16], we studied a more principled variant of this method, where we first preprocess the optimization problem, then perform randomized rounding and then, if necessary, correct any bad choices made during the rounding step. We call this more “intelligent” approach to randomized rounding combinatorial randomized rounding (CRR).

In CRR rounding, the most critical decisions that have to be taken are identified and then each critical decision is either nullified or a backup choice is provided for it. In order to be able to tolerate a number of backup choices after the rounding step, we preprocess the problems. We show that CRR, when it can be applied, achieves better approximation ratios than conventional randomized rounding. Moreover, all CRR-based randomized algorithms that we have derived can be derandomized by the so-called method of conditional probabilities. This gives us deterministic algorithms of an equivalent time complexity.

The main idea of the derandomization is as follows: in CRR-based algorithms we bound the probability that the randomly rounded solution violates the desired approximation ratio by a constant $r < 1$. This, in turn, proves the existence of at least one rounded solution that satisfies our approximation constraints. To get such a solution in a deterministic manner we build a decision tree, where on each level we assign the value 0 or 1 to an unknown integer.

To be able to make deterministic decisions, we then need an efficiently computable function $F$ that provides an upper bound on the probability of failure of an assignment for each node. We call such a function a pessimistic estimator. Derandomization along these lines therefore reduces to finding an appropriate pessimistic estimator. This idea goes back to work of Raghavan [31].

In [16], we manage to provide pessimistic estimators for all CRR-based algorithms that we propose, in particular for a wide class of scheduling and packing problems. Moreover, we could apply CRR in a uniform way to a variety of scheduling problems, where the processing time of a job may depend on the machine on which it will eventually be scheduled. In particular, we could handle the following objectives: minimizing the maximum load; finding a schedule of bounded makespan and cost, when costs are given together with processing times for any individual assignment; finding a schedule of bounded makespan with an arbitrary constant number of linear cost functions; maximizing the minimum load on any machine; minimizing the makespan when all machines must have almost equal load.

We also found pessimistic estimators for all these problems. From these we obtain fully polynomial approximations schemes (FPTAS) for the first four problems, and a polynomial time approximation scheme (PTAS) for the last. The FPTAS of the first problem has a linear time complexity. The FPTAS of the next three problems have an $O(n \log n)$ time complexity.
The last problem has a costly polynomial-time algorithm, due to a heavy-load preprocessing. In summary, our new CRR method together with the derandomization via pessimistic estimators, is simple to apply, uniform, and improves significantly on the performance of standard randomized rounding. For two particular problems, the resulting algorithms improve over previous work.

2.3 LP-rounding versus local search

In [17], we applied the LP-rounding technique described at the beginning of the previous section to the NP-hard problem of finding a maximum independent set in a very large graph. However, this approach could not be made to produce solutions that would fit the very strict quality demands of the application (a geometric packing problem from car industry).

We therefore also looked at the exact linear integer program formulation of the independent set problem. We checked whether we would we able to obtain optimal solutions for our problem sizes but that turned out to be infeasible. However, the ILP approach turned out to be useful in certain instances in that it could provide, in a reasonable time, lower and upper bounds on the value of an optimal solution value. Besides, we could use the ILP formulation to optimally solve small isolated subproblems.

The ultimately most successful approach turned out to be a simple local search heuristic, as it was proposed by Battiti and Protasi [7]. Their algorithm incrementally constructs an independent set, and in each step either adds or removes a vertex from that set. We give the main idea in the following paragraph, but for the problem of approximating the maximal clique, which is just the dual of the maximum independent set problem.

The main trick of the heuristic is to maintain, for each node, the number of nodes that could be added to the clique, if this node were removed from the clique. It can be shown that this information can be updated efficiently from a simple scan of a certain limited neighborhood of the (limited number of) nodes affected by an update. In practice, the algorithm will most of the times exchange only one single vertex for another until it manages to rearrange the local neighborhood to add one extra node. The algorithm keeps track of past steps to avoid cycles. The process is periodically restarted from scratch to avoid being trapped in a local maximum.

We also tried more sophisticated variants, with the algorithm trying to remove some \( n \) nodes in order to be potentially able to add \( n + 1 \) nodes. This, however, gave rise to infeasible ILP subproblems, so that the rather simple local search procedure from above ultimately gave us the best results.

3 Network Flows

3.1 Classical flow theory

Two of the most important concepts in classical flow theory are the concepts of preflows and blocking-flows. Goldberg and Tarjan [22] introduced PREFLOW-PUSH algorithms for the
maximum flow problem. Preflow-Push algorithms maintain at all times a so called pre-flow. A pre-flow is like a flow, except that the total amount flowing into a vertex is allowed to exceed the total amount flowing out. The pre-flow is changed gradually by pushing local flow excess towards the sink along edges that are estimated to be on shortest paths. Dinic’s algorithm [14] is a very popular blocking-flow algorithm. Such algorithms construct shortest path networks, called layered networks, and establish blocking-flows for this networks. In a blocking-flow any directed path from the source to the sink contains a saturated edge. Those basic, yet elegant techniques from classical flow theory have now their revival in the area of multiprocessor scheduling.

Scheduling unrelated parallel machines is among the most difficult scheduling problems. Independent jobs have to be assigned to a set of machines of different type without preemption. Processing job $i$ on machine $j$ takes time $p_{ij}$. For each machine $j$, the total time used by machine $j$ is the sum of processing times $p_{ij}$ for the jobs that are assigned to machine $j$. The makespan of a schedule is the maximum total time used by any machine. The objective is to find a schedule (assignment) that minimizes the makespan.

The unrelated scheduling problem is $NP$-complete, and it is known that unless $P = NP$, there is no polynomial-time approximation algorithm for the optimum schedule with approximation factor less than $\frac{3}{2}$. There exist polynomial-time algorithms with an approximation factor of $2$. These algorithms first compute an optimal fractional assignment and then use rounding to obtain a discrete solution. The optimal fractional assignment can be obtained by solving an LP (using interior point methods guarantees polynomial time), or combinatorially with the help of generalized network flows.

Many real-life large-scale optimization problems can be modeled as an unrelated scheduling problem. Typically, these problems arise in the area of scheduling multiprocessor computers and industrial manufacturing systems, where often groups of similar workstations having a variety of similar equipment with differing performance characteristics have to be optimized. For practical purposes, there have been proposed many heuristics solving the unrelated scheduling problem. They deliver both optimal and approximative solutions. Techniques used here range from combinatorial approaches with partial enumeration to integer programming with branch-and-bound and cutting planes.

The algorithm, we present in [20], computes an assignment for the unrelated scheduling problem with makespan at most twice the optimum. We prove that a 2-approximative schedule can be computed in $O(m^2A\log(m)\log(nU))$ time, where $A$ is the number of pairs $(i,j)$ with $p_{ij} \neq \infty$ and $U$ denotes the maximal $p_{ij} \neq \infty$. This is better than the previously known best time bounds. Our algorithm is linear in $A$. This is the first time that a combinatorial algorithm always beats the interior point approach for this problem. For problem instances where $A = \Theta(nm)$ our algorithm outperforms best known approaches by a factor almost linear in $n$.

Our algorithm is much simpler than the previously known algorithms. It always maintains an integral solution. It pushes unsplittable jobs through the two-layered bipartite network defined by the unrelated scheduling problem. It uses the primal-dual approach combined with a gain scaling technique to obtain a polynomial running time. To compute a flow among the
edges with zero reduced cost it uses the procedure Unsplittable-Blocking-Flow from [19] in the inner loop.

We feel that our approach has let to a much better understanding of the structure of the unrelated scheduling problem. A mixed continuous/discrete approach has been replaced by a complete discrete approach. Our algorithm takes advantage from addressing the approximation problem directly.

A special case of the unrelated scheduling problem is the restricted scheduling problem, in which each job $i$ has some weight $w_i$, each machine $j$ has some speed $s_j$ and $p_{ij} = \frac{w_i}{s_j}$ or $p_{ij} = \infty$ holds for all $i, j$. In [19], we design the basic procedure Unsplittable-Blocking-Flow and we give a combinatorial 2-approximation algorithm for the restricted scheduling problem. Our solution is based on the generic preflow-push algorithm. Moreover, for a simpler case with identical machines, our algorithm delivers an assignment at most $2 - \frac{1}{w_1}$ times worse than the optimal solution, where $w_1$ is the largest job weight. It improves on the best known approximation factor for this job scheduling problem.

Furthermore, for the restricted scheduling problem with identical machines, we present an algorithm which computes a pure Nash equilibrium from an arbitrary (not necessarily equilibrium) assignment without increasing the makespan. This technique is called Nashification. The algorithm combines the techniques from blocking flows with unsplittable flows. This is the first polynomial-time algorithm computing a Nash equilibrium for this problem.

### 3.2 Multiroute flows

For a long time, network flow theory has been used as a tool for many different optimization problems. Some of the most important applications deal with communication in a network. The communicating parties correspond to (some of) the nodes and the communication connections between pairs of nodes correspond to paths in the underlying network. Since classical flow is (roughly) a nonnegative linear combination of unit flows along paths, the multicommodity flow problem represents an appropriate tool. However, this framework is not always sufficient. One of the issues not addressed is the fault-tolerance. In the outlined model, a failure of a single link (edge) may cause interruption of several connections. The other ignored issue is the communication security. By listening to a single link (edge), an adversary may easily follow communication of other parties. These clearly undesirable features motivate the study of multiroute flows.

As already mentioned, a classical flow is a nonnegative linear combination of unit flows along simple paths. Multiroute flow generalizes this concept. The basic building blocks are not single paths with unit flows but rather tuples consisting of $h$ edge disjoint paths, each path carrying a unit flow. Such a structure of $h$ edge disjoint paths (from $s$ to $t$), each with a unit flow, is called an elementary $h$-flow. A multiroute flow (or a $h$-flow) is a nonnegative linear combination of elementary $h$-flows. Note that it only makes sense to talk of $h$-flow between pairs of vertices which are actually connected by at least $h$ edge disjoint paths. Whereas classical flow has the flexibility to split (and merge), $h$-flow has the obligation to split.

The idea of using several disjoint connections to ensure fault tolerance [24, 27] or security [15, 3]
is not new. However, most of the known algorithms for establishing multiple connections along disjoint paths for several pairs of nodes are heuristics, without any non-trivial worst case bounds (if there is just one pair of nodes, the Menger’s theorem solves the problem). To the best of our knowledge, the paper by Bagchi et al. [5] is the only one that gives a theoretical non-trivial result. The paper deals with an NP-complete integral variant of the multiroute multicommodity flow problem and describes an $O(h^3 \cdot F) = O(h^3 \alpha^{-1} \log n)$ approximation algorithm for it and for its variants (here $F$ denotes the flow number and $\alpha$ the expansion of the underlying network).

One of the results in the paper [4] is an improvement of the above mentioned bound on the approximation of integral multiroute multicommodity flow; we give an upper bound of $O(h^2 \cdot F) = O(h^2 \alpha^{-1} \log n)$. Moreover, we prove the same bound for a more general problem, for the coloring $h$-edge disjoint path systems which is a multiroute counterpart to the minimum path coloring problem [30]. The algorithms are based on rounding a fractional solution of a linear program. Typically, solving the linear program is the most time consuming part of rounding algorithms (cf. [34]). An interesting feature of our algorithms is that they avoid solving the linear program; instead, using a greedy-like procedure, they compute an approximate fractional solution and round the approximate solution.

The underlying problem is that of finding several disjoint path between a given pair of vertices. Menger’s theorem provides necessary and sufficient condition for existence of $h$ such paths. However, it does not say anything about the length of the paths whereas in communication problems the number of links used is an issue. We show that any two $h$-connected vertices are connected by $h$ edge disjoint paths of average length $O(hF)$.

An important yet not properly addressed problem is that of duality of multiroute flows and cuts. The problem was studied for single commodity multiroute flows and the celebrated duality of flows and cuts extends to this setting [26, 1], with a proper definition of the size of a cut. In the paper [6] we provide a direct elementary proof of this duality; we believe it provides a valuable new insight into the problem.

For classical multicommodity flows, an approximate duality of flows and cuts holds: the minimum cut is always at most $O(\log k)$ times larger than the maximum flow (with $k$ commodities), and in general, this is the best bound possible (cf. the survey by Shmoys [33]; in this short description we ignore the fact that there are at least two variants of the multicommodity flow problem, the approximate duality holds for both of them).

The situation gets much more complicated for multicommodity multiroute flows. The above mentioned result for multicommodity flow exploits the fact that a fractional solution to the minimum cut problem induces a metric on the underlying graph and the distances in this metric correspond to the lengths of shortest paths. In the case of multiroute flows, it is also possible to derive a metric from a fractional solution to the minimum cut problem yet the correspondence between distances and shortest paths is lost; as a result, the techniques applied in the proof of the approximate duality for the classical multicommodity flows and cuts do not work here. This area will be a subject of further research.
4 Non-Standard Objective Functions

4.1 I/O-efficient network optimization

Basic algorithmic research traditionally assumed some variant of the “von Neumann” model of computation with a single processor and uniform operation and memory access costs. However, more advanced models are by now an important established part of algorithmic research because many of the challenges in modern computer science have to do with communication, parallel computing, and memory hierarchies. In the following we will concentrate on shortest-path algorithms for memory hierarchies.

In connection with graph algorithms, the commonly accepted external-memory (EM) model defines some parameters: $N$ is the number of vertices and edges of the graph ($N = V + E$), $M$ ($< N$) is the number of vertices/edges that fit into internal memory, and $B$ is the number of vertices/edges that fit into a disk block. In an Input/Output operation (or I/O for short), one block of data is transferred between disk and internal memory. The measure of performance of an algorithm is the number of I/Os it performs. The number of I/Os needed to read $N$ contiguous items from disk is scan($N$) = $\Theta(N/B)$. The number of I/Os required to sort $N$ items is sort($N$) = $\Theta((N/B) \log_{M/B}(N/B))$ of $N$, $B$, and $M$, scan($N$) < sort($N$) $\ll N$. Advanced models include parallel disks (see Figure 1) or hide the parameters $M$ and $B$ from the algorithms (cache-oblivious model): due to their generality, cache-oblivious algorithms are efficient on each level of the memory hierarchy.

![Figure 1: The I/O model with parallel disks.](image)

External-memory graph-traversal algorithms like breadth-first search (BFS) and single-source shortest-path (SSSP) are still in their infancy. Until recently, I/O-efficient algorithms only existed for either dense graphs (which are quite rare in practice) or special (mostly undirected) graph classes like planar graphs. Recent work on external-memory graph traversal covered three major directions: (1) unifying approaches for undirected planar graphs, (2) tackling directed planar graphs, and (3) obtaining the first $o(V) + O(sort(V + E))$ I/O BFS algorithm for general undirected sparse graphs.

The efficiency of our BFS algorithm in (3) – and also of our recent follow-up results concerning SSSP, APSP [2], and cache-oblivious BFS/SSSP [12] – stems from the fact that the graphs we
consider are undirected; one exploits that whenever the first node of a cluster is visited then the remaining nodes of this cluster will be reached soon after. By spending only one random I/O in order to load the whole cluster and then keeping the cluster data in some efficiently accessible data structure until it is all used up, the total amount of I/O can be reduced. On sparse graphs, essentially a factor of $\sqrt{B}$ can be gained by this technique.

In our APSP paper [2] we show how to gain yet another factor of $\sqrt{B}$ (and thus achieving I/O-optimality) by solving all $V$ SSSP problems concurrently. While this is a nice result from a theoretical point of view, blowing up the external work space by a factor of $V$ may be infeasible. Therefore we also explored how to reduce the work space while still maintaining I/O-efficiency. For example, in the case of all-pairs BFS it is sufficient restrict concurrency to subproblems with sources in the same cluster: since the resulting BFS trees are quite similar in shape the I/O access sequence for one subproblem can be reused to solve the others, too.

In [12] we present improved cache-oblivious data structures and algorithms for BFS and SSSP on undirected graphs with non-negative edge weights. Our results close the performance gap between the currently best external-memory algorithms for these problems and their cache-oblivious counterparts. Our shortest-path algorithm relies on a new data structure, called bucket heap, which is the first cache-oblivious priority queue to efficiently support a weak DECREASEKEY operation.

4.2 Average-case and smoothed analysis

Most theoretical analyses of algorithms are worst-case in the sense that they provide bounds on the worst possible performance of an algorithm. On the one hand, this provides a strong guarantee as it cannot be obtained from any experimental study. On the other hand, a single upper bound cannot truly reflect the overall performance of an algorithm, especially in cases where that performance happens to vary widely from instance to instance.

A quite extreme example in that respect is the knapsack problem. All known algorithms have a worst-case time complexity that is exponential in the problem size (that is, essentially, the number of items to be packed). However, if one picks an instance uniformly at random then the average running time grows only linearly in the size of the instance. The reason for this phenomenon is that almost all knapsack instances are easy to solve such that a randomly chosen instance is most likely an easy one. This shows that for a comprehensive evaluation of algorithms, worst-case analysis is of only limited use.

A quite natural approach to remedy this deficit is to consider the average performance of algorithms over all instances which is the underlying idea of average-case analysis. More precisely, given a probability distribution on the set of instances, the performance of an algorithm becomes a random variable which is subsequently investigated, for example, by bounding its expectation. Although average-case analysis can reveal a lot about the performance of an algorithm or even the complexity of the problem at hand, it is hard to draw sensible conclusions for the performance in practice. The reason is that real world instances are not totally random but usually have some special structure and properties.

A new measure is provided by smoothed analysis which combines elements from worst case
and average-case analysis. As in worst-case analysis, the maximum over all instances is taken. But instead of directly measuring the performance of each individual instance, the expected performance under a small random perturbation of the original instance is considered. The performance is then measured with respect to the size of the instance and the magnitude of the perturbation, described by some parameter $\sigma$. For $\sigma \rightarrow 0$, the original instance is not perturbed at all which corresponds to the standard worst-case analysis. For very large $\sigma$, the random perturbation swamps out the adversarial input values of the original instance and one obtains average-case analysis. Using smoothed analysis we can describe the transition between these two extremes. Smoothed analysis upper-bounds the average performance on instances in a parameterized neighborhood of the worst-case instances. For algorithms with about equal performance on all problem instances that have the same size, worst-case, average-case and smoothed analysis should yield the same result. Assume the worst-case and average-case measure differ significantly. If the worst-case instances are clustered together, then also the smoothed performance is bad. If, however, the worst-case instances are “isolated events” in instance space one would expect a rapid improvement of the running time bound with an increasing size of the neighborhood. In other words, if the smoothed measure is low, worst case instances are not robust under small changes.

Which problems allow fast algorithms under small perturbations? This question leads to a new field in complexity theory. Recall that the worst-case measure is the basis for the standard complexity theory. We define a class of efficiently solvable problems under small perturbations which consists of all problems that have polynomial smoothed complexity, an adaption of polynomial average-case complexity with the additional requirement that the running time has to be polynomially bounded not only in the size of the problem instance but also in the magnitude of the perturbation. This complexity measure is slightly weaker than expected polynomial running time, but is a more robust measure, as it is independent of the used machine model.

Many combinatorial optimization problems have an objective function or constraints specified in terms of real numbers representing natural quantities like time, weight, distance, or utility. This includes some well-studied large-scale optimization problems from network design and routing like, e.g., traveling salesperson, shortest path, minimum spanning tree as well as various scheduling and packing problems. When analyzing the complexity of algorithms for such problems, we usually assume that these numbers are integers or rational numbers with a finite length representation. The hope is that it suffices to measure and compute with some bounded precision in order to identify an optimal or close to optimal solution. In fact, if real numbers occur only in the objective function and if this objective function is well-behaved (e.g., a linear function) then calculating with reasonable approximations of the input numbers yields a feasible solution whose objective value is at least close to the optimal objective value. More problematically, however, if the constraints are defined by real numbers, then calculating with rounded input numbers might miss all interesting solutions or might even produce infeasible solutions.

How can one solve optimization problems (efficiently) on a computer when not even the input numbers can be specified exactly? – In practice, optimization problems in which real numbers
occur in the input are solved by simply rounding the real numbers more or less carefully. Fortunately, this approach seems to yield reasonable results. We seek for a theoretically founded explanation why this rounding approach usually works. Studying this issue under worst-case assumptions does not make very much sense as, in the worst case, the smallest inaccuracy might lead to an infeasible or utterly sub-optimal solution. This question needs to be studied in a stochastic model. We present a probabilistic analysis for a very large class of combinatorial optimization problems containing, e.g., all binary optimization problems defined by linear constraints and a linear objective function over \( \{0,1\}^n \), but also problems with a non-linear feasible region [8]. By parameterizing which constraints are of stochastic and which are of adversarial nature, we obtain a semi-random input model that enables us to do a general average-case analysis for a large class of optimization problems while at the same time taking care of the combinatorial structure of individual problems. Our analysis covers various probability distributions for the choice of the stochastic numbers and includes smoothed analysis with Gaussian and other kinds of perturbation models as a special case. In particular, the feasible region can be described by an arbitrary set \( S \subseteq \{0,1\}^n \) intersected by some stochastic constraints, where the set \( S \) is assumed to be specified by an adversary in an arbitrary possibly non-linear and non-convex way. Only the stochastic constraints are assumed to be linear. In this semi-random input model, we can exactly characterize the smoothed complexity of optimization problems in terms of their random worst-case complexity.

**Theorem:** A binary optimization problem has a polynomial smoothed complexity if and only if it has a pseudo-polynomial complexity.

Our analysis is centered around structural properties of binary optimization problems, called winner, loser, and feasibility gaps. We show that when the coefficients of the objective function and/or some of the constraints are stochastic, then there usually exist a polynomial \( n^{-\Omega(1)} \) gap between the best and the second best solution as well as a polynomial slack to the boundary of the constraints. Similar to the condition number for linear programming, these gaps describe the sensitivity of the optimal solution to slight perturbations of the input and can be used to bound the necessary accuracy as well as the complexity for solving an instance. Our analysis shows that one can usually round real-valued input numbers after only a logarithmic number of bits without changing the optimal solution. In fact, our probabilistic analysis goes far beyond the point of explaining phenomena occurring in practice. We are able to provide algorithms with polynomial average-case complexity (more precisely, polynomial smoothed complexity) for a quite general class of discrete optimization problems. Technically, we exploit the gap properties in form of an adaptive rounding scheme increasing the accuracy of calculation until the optimal solution is found. The strength of our techniques is illustrated by applications to various NP-hard optimization problems from mathematical programming, network design, and scheduling for which we obtain the first algorithms with polynomial average-case/smoothed complexity. For example, we obtain algorithms with polynomial smoothed complexity for the multi-dimensional knapsack problem, the constrained spanning tree problem, the constrained shortest path problem, and the problem of scheduling to minimize the weighted number of tardy jobs.

In a related line of research we experimentally investigate structural properties of knapsack instances under a probabilistic setting and present algorithms that specifically exploit those
structural properties [8]. In particular, we are interested in the average number of Pareto-optimal knapsack fillings. Pareto optimality is a generalization of optimality from single-criteria to multi-criteria optimization. Translated to the knapsack problem, a solution is Pareto optimal if there exists no other solution that is better with respect to weight and profit at the same time. While in the worst-case, the number of Pareto optimal knapsack fillings is exponential in the number of items, we observe only a quadratic growth when either the weights or the profits of the items contains a sufficient amount of randomness. Pareto-optimal knapsack filling can be enumerated very efficiently. Enumerating all Pareto optimal knapsack fillings implicitly solves also the the original knapsack problem. This way we obtain algorithms which, on average, exhibit a polynomial running time for random knapsack instances. We present even faster algorithms which additionally exploit the fact that the average integrality gap is very small for random knapsack instances. In our experiments these algorithms show a running time that is linear in the number of available items provided that weights and profits are chosen uniformly at random.

The common approach in the theoretical and the experimental study is the analysis of certain structural properties of optimization problems under a probabilistic setting. We use our finding to develop algorithms that, by specifically exploiting these properties, are very efficient for most of the problem instances.

References


[23] Gondzio J. and Sarkissian, R., Column Generation with a Primal-Dual Method, Logi-
lab Technical Report 96.6, Department of Management Studies, University of Geneva, Switzerland.

connections using maximum 2-route flows, IEEE/ACM Transactions on Networking 11:5,


[26] Kishimoto, W., A Method for Obtaining the Maximum Multiroute Flows in a Network,

[27] Lee, S.-J. and Gerla, M., Split Multipath Routing with Maximally Disjoint Paths in Ad
hoc Networks, in Proceedings of the IEEE International Conference on Communications

[28] Luby, M. and Nisan, N., A parallel approximation algorithm for positive linear program-
ing, in Proceedings of the 25th ACM Symposium on Theory of Computing (STOC’93),


[31] Raghavan, P., Probabilistic construction of deterministic algorithms: Approximating
packing integer programs, Journal of Computer and System Sciences 37, pp. 130–143,

[32] Raghavan, P. and Thompson, C. D., Randomized rounding: a technique for provably

[33] Shmoys, D. B., Cut Problems and Their Application to Divide-and-Conquer, Approxima-

[34] Young, N. E., Randomized Rounding without Solving the Linear Program, in Proceedings

[35] Young, N. E., Sequential and parallel algorithms for mixed packing and covering, in Pro-
cedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS’01),