Introduction

Sets and elements

Any type of objects can be viewed as elements of an abstract set. We can talk about sets of integers, sets of real numbers, sets of colors, or sets of buildings, etc. For a set Q and element q we write $q \in Q$ if q belongs to the set Q, and we write $q \notin Q$ otherwise. For two sets S and Q we can consider their union $S \cup Q$ consisting all elements that belong to S or Q (or both). Their intersection $S \cap Q$ is the set of elements that belong to both S and Q. It maybe the empty set which we denote by $\emptyset$. For a set S we denote by $|S|$ its cardinality, that is the number of its elements. For some sets this maybe infinite (or even uncountable). If V is an abstract set of certain type of objects and Π is a property, we can define a subset of V by $S = \{v \in V \mid v \text{ has property } \Pi\}$, which is the set of (all) elements of V that has property Π. Using this notation, we can write for instance $S \cap Q = \{i \in S \mid i \in Q\}$. For two sets S and Q we define their difference set as $S \setminus Q = \{i \in S \mid i \notin Q\}$.

Sometimes we assign real (or integer) weights w to a finite set V which we view as a mapping: $w : V \mapsto \mathbb{R}$. In this case for $v \in V$ we denote by $w(v)$ the associated weight (length, cost, etc.). For a subset $S \subseteq V$ we use $w(S) = \sum_{v \in S} w(v)$. Sometimes we view the same weight assignment as a vector $w \in \mathbb{R}^V$. In this case we denote by w_v the weight assigned to $v \in V$. We still use $w(S) = \sum_{v \in S} w_v$ for the sum of the weights over a subset.

Vectors and matrices

We denote by $\mathbb{R}$ the set of reals numbers, by $\mathbb{Z}$ the set of integers, and by $\mathbb{Z}_+$ the set of nonnegative integers. For an abstract set V we denote by $\mathbb{R}^V$ the set of real vectors that are indexed by the set V. In other words, $x \in \mathbb{R}^V$, then its components are x_v, for $v \in V$. Sometimes we write $x = (x_v \mid v \in V)$. We also say that the set V is the index set of vector $x \in \mathbb{R}^V$.

For a nonnegative integer $n \in \mathbb{Z}_+$ we use $[n] = \{1, 2, ..., n\}$ to denote the set of positive integers not larger than n. To simplify notation we write $\mathbb{R}^n$ instead of $\mathbb{R}^{[n]}$. Thus, $\mathbb{R}^n$ is the usual Euclidean space of dimension n.

For two sets R and C we denote by $\mathbb{R}^{R \times C}$ the set of matrices the rows of

which are indexed by the elements of R and the columns of which are indexed by the elements of C. For such a matrix $A \in \mathbb{R}^{R\times C}$ we denote by $a_{r,c}$ its components for $r \in R$ and $c \in C$. For a matrix $A \in \mathbb{R}^{R\times C}$ and vector $x \in \mathbb{R}^C$ we define $y = Ax \in \mathbb{R}^R$ by $y_r = \sum_{c\in C} a_{r,c} \cdot x_c$ for all $r \in R$. In particular, we can view a vector $x \in \mathbb{R}^V$ as a $V \times 1$ matrix (that is $\mathbb{R}^{V\times[1]} = \mathbb{R}^V$), and we will write x^T if we want to view it as a $1 \times V$ matrix. Sometimes we say that x is a column vector, and x^T is a row vector. For two vectors $x, y \in \mathbb{R}^V$ thus we can interpret x^T as a $1 \times V$ matrix, and hence write $x^T y$ to denote the resulting 1-dimensional vector, that is $x^T y = \sum_{i\in V} x_i \cdot y_i \in RR$. It is called the scalar product of the two vectors. Note that for $A \in \mathbb{R}^{R\times C}$ and $x \in \mathbb{R}^Q$ for some set $Q \neq C$ the expression Ax does not make any sense. For a matrix A and vector x the expression Ax makes only sense if the index set of the columns of A is the same as the index set of x. Similarly, for a vector y the expression $y^T A$ makes only sense if the index set of y is the same as the index set of the rows of A, in which case $y^T A = x^T$ is a row vector defined by $x_c = \sum_{r\in R} y_r \cdot a_{r,c}$ for $c \in C$.

For the particular case when $R = [m]$ and $C = [n]$ for some positive integers m and n we simply write $A \in \mathbb{R}^{m\times n}$ to indicate that A is a real matrix with rows indexed by $i = 1, ..., m$ and columns indexed by $j = 1, ..., n$.

For a set V we write 1_V to denote the vector in $\mathbb{R}^V$ in which all components are equal to 1. Similarly, we denote by 0_V the full 0 vector of $\mathbb{R}^V$. For the case of $V = [n]$, we simply write 1_n and 0_n.

For vectors $x, y \in \mathbb{R}^n$ we write $x \leq y$ if we have $x_j \leq y_j$ for all indices $j = 1, ..., n$.

Graphs

A simple graph is the pair $G = (V, E)$ where V is a finite set and $E \subseteq V \times V$. If we interpret E as a collection of unordered pairs then G is an undirected graph, while if we view E as a set of ordered pairs then G is a directed graph. We call the elements of V vertices, and the elements of E edges. If G is a directed graph then we call its edges also as arcs. Sometimes we use $G = (V, A)$ for a directed graph to emphasize that its edges are ordered pairs, called arcs. If $E = (u, v) \in E$ for a graph $G = (V, E)$ then we say that edge e is incident with vertices u and v. If G is directed then we call vertex u the

tail of e and vertex v the head of e.

To an undirected graph we associate its incidence matrix $A \in \mathbb{R}^{E\times V}$. IN fact A will contain only 0 or 1 entries, thus we could write $A \in \{0,1\}^{E\times V}$. The rows of A are indexed by the edges of G and the columns of A are indexed by the vertices of A. The components of A are defined by $a_{e,v} = 1$ if and only if (sometimes we shorten this to iff) edge e is incident with vertex v (and otherwise it is 0, since we assumed that $A \in \{0,1\}^{E\times V}$. This means that each row of A contains exactly two nonzero entries, in the columns corresponding to the vertices of the edge indexing this row. The incidence matrix A can be viewed as a representation of G.

We call a simple graph $G = (V, E)$ bipartite if $V = A \cup B$, $A \cap B = \emptyset$ and $E \subseteq A \times B$. In other words if the vertex set is partitioned into two subsets and all edges are going between these two subsets. We usually write $G = (A, B, E)$ to indicate that g is a bipartite graph with vertex partition $V = A \cup B$. We can also represent such a bipartite graph by its vertex-vertex adjacency matrix $A \in \{0,1\}^{A\times B}$. Here rows are indexed by vertex set A and columns are indexed by vertex set B, and we have $a_{u,v} = 1$ iff $(u, v) \in E$.

A directed graph $G = (V, E)$ can also be represented by its incidence matrix, which is defined as follows. $A \in \{-1, 0, +1\}^{E\times V}$, where

$$a_{e,v} = \begin{cases} -1 & \text{if } e = (v, w) \text{ for some vertex } w, \\ +1 & \text{if } e = (u, v) \text{ for some vertex } u, \\ 0 & \text{otherwise.} \end{cases}$$

Given a simple undirected graph $G = (V, E)$ we denote by $N(v)$ the neighbors of vertex v, that is $N(v) = \{u \in V \mid (u, v) \in E\}$, $v \in V$. The degree of vertex v is the number of its neighbors, that is $d_G(v) = |N(v)|$.

For a directed graph $G = (V, E)$ we denote by $N^+(v)$ the set of out-neighbors of v, that is $N^+(v) = \{w \in V \mid (v, w) \in E\}$, and denote by $d^+_G(v) = |N^+(v)|$ its out-degree. Similarly, $N^-(v) = \{u \in V \mid (u, v) \in E\}$ is the in-neighborhood of v, and $d^+_G(v) = |N^-(v)|$ is its in-degree.

Undirected and directed graphs can be used as model for many pairwise relations arising in practical problems. Graphs will arise as natural models in many of the problem we will consider in this course. Further simple graph theoretical notions will be introduced during later as needed.

Binary vectors and the Boolean cube

Given a finite set V of cardinality $|V| = n$, we can represent a subset $S \subseteq V$ by its characteristic vector $\chi(S) \in \{0,1\}^V$ defined by $\chi(S)_v = 1$ iff $v \in S$ (and it is 0 otherwise). We denote by $\mathbb{B} = \{0,1\}$, and by $\mathbb{B}^V = \{0,1\}^V$ the so called Boolean cube, that is the set of binary vectors of dimension $n = |V|$. To a binary vector $x \in \mathbb{B}^V$ we can also associate a subset of V by defining $\mathit{support}(x) = \{v \in V \mid x_v = 1\}$. It is easy to see that these two definitions map subsets of V in a one-to-one way to $\mathbb{B}^V = \{0,1\}^V$. Namely, we have $\chi(\mathit{support}(x)) = x$ for all binary vectors $x \in \mathbb{B}^V$.

Hypergraphs

Traditionally we denote the set of all subsets of V by 2^V (and call it the power set of V) to emphasize the above correspondence to binary vectors and the fact that $|2^V| = 2^{|V|}$. A hypergraph is simply a subfamily of the power set of V, that is $\mathcal{H} \subseteq 2^V$ is a hypergraph on vertex set V. We call a subset $H \in \mathcal{H}$ an edge (or hyperedge) of $\mathcal{H}$.

The degree of a vertex in a hypergraph $\mathcal{H}$ is the number of hyperedges containing the vertex, that is $d_{\mathcal{H}}(v) = |\{H \in \mathcal{H} \mid v \in H\}|$. We call a hypergraph k-uniform if for all hyperedges $H \in \mathcal{H}$ we have $|H| = k$. The dimension of a hypergraph is the size of its largest edge, that is $\dim(\mathcal{H}) = \max\{|H| \mid H \in \mathcal{H}\}$.

Hypergraphs can be considered simple generalizations of graphs, though for historical reasons we use different notation. Given a graph $G = (V, E)$, its edge set is a 2-uniform hypergraph on vertex set V. Conversely, every 2-uniform hypergraph $E \subseteq 2^V$ on vertex set V defines a graph $G = (V, E)$.

Similarly to graphs, hypergraphs can also be represented by their edge-vertex incidence matrix $A \in \mathbb{B}^{\mathcal{H} \times V}$. The rows of matrix A are indexed by the hyperedges $H \in \mathcal{H}$, while the columns are indexed by the vertices $v \in V$. The matrix is defined by $a_{H,v} = 1$ iff $v \in H$ for all $H \in \mathcal{H}$ and $v \in V$.

Complexity of algorithms

We will need to evaluate the efficiency of algorithms, and compare different algorithms. The two general principle is that such an evaluation should be *reasonable* and should work *in worst* case, too. To be able to explain these principles and we need to recall some standard notation.

Assume $f(n)$ and $g(n)$ are positive functions over the set $\mathbb{Z}_+$ of nonnegative integers. We write $g(n) = O(f(n))$ if there exists a constant c and threshold $n_0 \in \mathbb{Z}_+$ such that we have $g(n) \leq c \cdot f(n)$ for all integers $n \geq n_0$. We write $g(n) = o(f(n))$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$. Finally we write $g(n) = \Omega(f(n))$ iff $f(n) = O(g(n))$.

The reasonable evaluation of algorithmic efficiency means that we view the number of operations needed by an algorithm for solving a problem as a function of the input size, and view the function itself as a statement about the efficiency of the algorithm. More precisely, a typical algorithm is designed to solve a certain type of problems. Say algorithm $\mathfrak{A}$ is designed solving problems in the problem class (set) $\mathcal{P}$, and we assume that $\mathcal{P}$ contains infinitely many different problem instances, that may have arbitrary sizes. For an instance $I \in \mathcal{P}$ we say its binary size $size(I)$ is essentially the number of bits one needs to use on a digital computer to represent the particular problem I. Thus, if I involves some integer numbers, say $a \in \mathbb{Z}_+$, then we use $\lceil \log_2 a \rceil$ bits to represent this number. If I involves several integer parameters, then its binary size $size(I)$ is the sum of the binary sizes of these parameters. We will see some examples for certain types of problems and the computation of their sizes.

When we evaluate the efficiency of algorithm $\mathfrak{A}$ then the reasonable evaluation implies that we view the number of arithmetical/logical/storage/comparative steps $length(\mathfrak{A}, I)$ that algorithm $\mathfrak{A}$ has to do when solving problem I as a function of $size(I)$. The principle of worst case evaluation means that we are seeking a function $f(n)$ such that $length(\mathfrak{A}, I) = O(f(size(I)))$ holds. In other words, apart from some small instances, the running time (the number of operations) $\mathfrak{A}$ makes when solving an instance $I \in \mathcal{P}$ can always be bounded by $c \cdot f(size(I))$.

In particular we say that $\mathfrak{A}$ is a linear algorithm if $length(\mathfrak{A}, I) = O(size(I))$, and it is a polynomial algorithm if $length(\mathfrak{A}, I) = O(size(I)^k)$ for some integer k that is independent of instance I.

For practical purpose, we also consider the unary size of problem instances. The unary size of a positive integer a is $usize(a) = a$. If a problem instance I involves several integer parameters, the its unary size $usize(I)$ is the sum of the unary sizes of its parameters.

Note that the unary size is typically exponentially larger than the binary size. For instance considering a single positive integer a, it is easy to see that for any integer k we have $usize(a) \neq O(size(a)^k)$. However for some small

enough integers a the unary size $usize(a)$ and the binary size $size(a)$ are not that far.

We say that an algorithm $\mathfrak{A}$ is pseudo-polynomial for problem family $\mathcal{P}$ if $length(\mathfrak{A}, I) = O(usize(I)^k)$ for some integer k. This means that while for some problems $I \in \mathcal{P}$ algorithm $\mathfrak{A}$ may need exponential amount of time in terms of $size(I)$ to solve instance I, for some other instances it may only need polynomial time (for those where all integer parameters are smaller than a fixed constant).

Complexity of problem classes

When we evaluate a type of problems as "hard" or "easy" we rely on a classification introduced in the early 1970-ies. Today this area is known as complexity theory, and it grew into a very large area. For this course we need to know only a few notions; we are not going to engage in complicated reductions or proofs of complexity.

First thing to remember that this complexity classification we use classifies problem classes (that is infinite families of problem of the "same" type) and not problem instances. A particular problem instance does not have a complexity.

Second things is that this classification is only for decision problems, that is for problems where the output is either a "YES" or a "NO". We shall see in class that all optimization problems we consider in this course can be reduced to a series of decision question (in polynomial time). Thus, we shall use for the optimization problems the same classification as for their decision variant.

The third thing to note is that the classification is based on the notion of (polynomial) reduction. We say that a problem class $\mathcal{A}$ is reducible to another problem class $\mathcal{B}$ if there exists integers k and ℓ and an algorithm $\mathfrak{A}$ that solves every instance $I \in \mathcal{A}$ in $O(size(I)^k)$ time by using $O(size(I)^\ell)$ calls to algorithm $\mathfrak{B}$, where $\mathfrak{B}$ is an algorithm that can solve any instance of $\mathcal{B}$ in unit time. Note that for such a reduction we do not need to know algorithm $\mathfrak{B}$, it is enough to assume its existence. We only use its output in $\mathfrak{A}$. If $\mathcal{A}$ is reducible to $\mathcal{B}$, then we write $\mathcal{A} < \mathcal{B}$.

A fourth thing to note is the notion of verification. We say that for a problem instance $I \in \mathcal{A}$ the "YES" answer if verifiable if there exists a polynomial time (in terms of the size of I) proof that indeed "YES" is the correct answer. Note that this is very different from being able to find the

correct answer for I in polynomial time. If for all instances $I \in \mathcal{A}$ the "YES" is verifiable, then we say that $\mathcal{A}$ belongs to NP. Similarly, if for instances $I \in \mathcal{A}$ the "NO" is verifiable, then we say that $\mathcal{A}$ belongs to $co-NP$.

A fifth thing is the actual classifications. We say that a problem class $\mathcal{B}$ is NP-complete, if it belongs to NP and for all other problem classes $\mathcal{A} \in NP$ we have $\mathcal{A} < \mathcal{B}$. We say that a problem class $\mathcal{B}$ is NP-hard if for all problem classes $\mathcal{A} \in NP$ we have $\mathcal{A} < \mathcal{B}$. Note that the reduction relation is transitive, that is if $\mathcal{A} < \mathcal{B}$ and $\mathcal{B} < \mathcal{C}$ then we also have $\mathcal{A} < \mathcal{C}$. Thus, a problem class $\mathcal{B}$ is NP-hard if there exists a single NP-complete problem class $\mathcal{A}$ such that $\mathcal{A} < \mathcal{B}$. If at the same time $\mathcal{B} \in NP$ then it is also NP-complete. Similarly we talk about $co-NP$-complete and $co-NP$-hard problem classes. Finally we say a a problem class $\mathcal{A}$ is polynomial, if there exists and algorithm $\mathfrak{A}$ and an integer k such that for all problem instances $I \in \mathcal{A}$ algorithm $\mathfrak{A}$ solves I in $O(size(I)^k)$ time. We denote by P the family of polynomially solvable problem classes.

The last thing to remember is that problems $\mathcal{A} \in NP \cap co-NP$ are called well-characterized and are expected to be solvable efficiently. In particular, we know that $P \subseteq NP \cap co-NP$, and here equality is a longstanding open conjecture.

Convexity

Convexity is a fundamental notion used in all branches of optimization theory. Given d-dimensional vectors $v^j \in \mathbb{R}^d$, $j = 1, ..., n$ we say that

$$x = \sum_{j=1}^{n} \lambda_j v^j \tag{1}$$

is a *linear combination* of v^1, v^2, ..., v^n if $\lambda_j \in \mathbb{R}$ for all $j = 1, ..., n$. If we have $\lambda_j \geq 0$ for all $j = 1, ..., n$ the we call x a *nonnegative combination* or a *conical combination* of vectors v^1, v^2, ..., v^n. Finally, if $\sum_{j=1}^{n} \lambda_j = 1$ and $\lambda_j \geq 0$ for all $j = 1, ..., n$, then we call x a *convex combination* of the vectors v^1, v^2, ..., v^n.

The *convex hull* of given vectors is defined as the set of all convex combinations of these vectors.

$$\text{conv}\left(\{v^1, v^2, ..., v^n\}\right) = \left\{ \sum_{j=1}^{n} \lambda_j v^j \,\middle|\, \sum_{j=1}^{n} \lambda_j = 1,\ \lambda_j \in \mathbb{R}^+,\ j = 1, ..., n \right\}$$

The convex hull of finite number of vectors is also called a *polytope*.

A subset $K \subseteq \mathbb{R}^d$ is called a *convex set* if for all $x, y \in K$ we have $\text{conv}(\{x, y\}) \subseteq K$. It is called a *polyhedron* (or polyhedral convex set) if there exist a positive integer m, a matrix $A \in \mathbb{R}^{m \times d}$ and a vector $b \in \mathbb{R}^m$ such that

$$K = \{x \in \mathbb{R}^d \mid Ax \leq b\}.$$

In other words, a polyhedron is the set of *feasible solutions* to a finite set of linear inequalities. Note that K may not be bounded. If it is bounded then it is a polytope, that is it is also the convex hull of finitely many points. In other words, there exists a subset $V \subseteq K$, $|V|$ finite such that

$$K = \text{conv}(V).$$

Such a subset is not unique, but they all contain a unique minimal set V (containmentwise minimal). This minimal set V is the set of *vertices* of the polytope K.

Another characterization of a vertex v of a polytope K is that there exists no $x, y \in K$, $x \neq v \neq y$ such that $v \in \text{conv}(\{x, y\})$.

Linear programming

Linear programming is a fundamental tool in optimization in general, and also in discrete optimization.

Given a matrix $A \in \mathbb{R}^{m \times n}$, and vectors $B \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ a linear program is the optimization problem that can be written as

$$\begin{array}{lrcl} \max & c^T x & & \\ s.t. & & & \\ & Ax & \leq & b \\ & x & \geq & 0_n \end{array} \tag{2}$$

Note that the set of feasible solutions

$$K = \{x \in \mathbb{R}^n \mid Ax \leq b,\ x \geq 0_n\}$$

is a convex polyhedral set. It maybe unbounded.

If $K = \emptyset$, then we say that problem (2) is *infeasible*.

If $K \neq \emptyset$ and there exists a vector $\hat{x} \in \mathbb{R}^n$ such that $c^T \hat{x} > 0$ and $A\hat{x} \leq 0_m$, then we call problem (2) *unbounded*. Note that in this case for any $\alpha > 0$

and $x \in K$ we have $x + \alpha \cdot \hat{x} \in K$, that is K is not bounded. Furthermore, we have

$$\lim_{\alpha \to +\infty} c^T(x + \alpha \cdot \hat{x}) \;=\; +\infty.$$

Finally, if none of the above, then the fundamental theorem of linear programming states that problem (2) has a finite optimum $x^* \in K$ (may not be unique). That is one for which we have $c^T x \leq c^T x^*$ for all $x \in K$.

Note next that problem (2) has a co-joined twin brother; even if it is not "visible" on input, it is still there. We call (2) the *primal* problem, and its co-joined twin its *dual*. The dual of (2) and can stated as

$$\begin{array}{lrcl} \min & y^T b & & \\ s.t. & & & \\ & y^T A & \geq & c^T \\ & y & \geq & 0_m \end{array} \tag{3}$$

Note first that the dual is defined by the same input A, b and c, and is obtained simply by interchanging the roles of constraints and variables in the primal. Note also the (3) can be brought into the form of the primal by simple equivalence transformations. This implies that (3) itself is a linear program. If we apply the same variable-constraint interchange as we used to obtain the dual from the primal to this equivalent form of (3) then we get a problem that is equivalent to (2). In other words the primal and dual problems form a pair of problems that are dual to one another.

We should remember two fundamental theorems that relate these two problems.

Theorem 1 (Weak Duality) *If x is feasible in the primal and y is feasible in the dual then we have*

$$c^T x \;\leq\; y^T b.$$

Theorem 2 (Strong Duality) *If any one of these problems has a finite optimum then the other one also has a finite optimum. Furthermore, if x^* is optimal to the primal and y^* is optimal to the dual, then we have*

$$c^T x^* \;=\; (y^*)^T b.$$