Turing and Mapping Reducibility

Last Updated: April 15th, 2024

1 Turing Reducibility

A common technique in everyday problem solving is to leverage the solution to one problem in order to solve another. For example, consider our friend Sam who must solve the problem of earning enough money to help support his way through college. One possible solution that Sam has considered is to work part-time delivering food for DoorDash. This solution would leverage the fact that he's already solved the problem of driving a vehicle from any one city location to another. Sam will likely have to solve tens of transport problems during a single work shift. Furthermore, the solution to the problem of successful navigation during a single instance of the food-transport problem can in be reduced to the problem of querying a global positioning system about one's position on earth at any moment during the trip.

In computer science, when an algorithm solves an instance of problem A by making one or more queries about the solutions to instances of problem B, then we say that A is *Turing reducible* to B, named after the British mathematician Alan Turing (1912-1954) who provided one of the earliest known theoretical models of computation that is functionally equivalent to the computers we use today (so long as our computers are idealized as having an infinite supply of memory). In this lecture we take a closer look at Turing reducibility and how it can be used as a means for devising algorithms.

Turing reducibility represents an important tool for desigining an algorithm to solve some problem A by leveraging an existing algorithm that solves another problem B by calling B's algorithm one or more times for different instances of B in order to solve a single instance of A.

Example 1.1. Consider the problem of multiplying two positive numbers, say 5 and 3. Marcia is still learning the multiplication table, but she performs well in addition, and also knows that 5×3 means $(5 + 5) + 5$. Thus, she first solves the addition problem $5 + 5$ and gets the answer 10. She then solves the final addition problem, $10 + 5$ to obtain the answer of 15.

The following function returns the product of its two inputs by making calls to an `add` function that returns the sum of its two inputs. It essentially generalizes Marcia's solving method.

```
unsigned int multiply(unsigned int a, unsigned int b)
{
    int sum = 0;
    int i;

    for(i=1; i <= b; i = add(i,1))
        sum = add(sum,a);

    return sum;
}
```

In Example 1.1, Marcia *reduced* the `Multiply` problem to the `Add` problem. In other words, she devised an algorithm for multiplying two numbers that relies on solving one or more addition problems. Moreover, whenever the answer to an instance of `Add` is being sought to help solve an instance of `Multiply`, then we say that `Multiply` is making a **query** to the `Add`-**oracle**, i.e., an entity that is capable of providing solutions to instances of `Add`. The answer provided by the oracle is called a **query answer**. Note that the algorithm that is making queries to an oracle does not necessarily need to know how the oracle is providing its answers. In case of the `multiply` function in Example 1.1, the function just assumes that each `add` query will be correctly answered with no concern about how the answer is obtained. In fact, the oracle may provide answers to instances of a problem for which it is impossible to devise an algorithm for solving it.

Definition 1.2. Problem A is **Turing reducible** to problem B, denoted $A \leq_{\mathrm{T}} B$, iff there is some algorithm that can solve any instance x of A, and is allowed to make zero or more queries to a B-oracle, i.e. an oracle that provides solutions to instances of B.

Definition 1.3. If $A \leq_{\mathrm{T}} B$ via an algorithm whose running time $\mathrm{O}(n^k)$, for some $k > 0$, then A is said to be **polynomial-time Turing reducible** to B, denoted $A \leq_{\mathrm{T}}^{\mathrm{p}} B$. Note: this definition assumes that each B-query is answered in one step.

Interesting fact: the term *oracle* comes from ancient Greece, where the "oracle at Delphi" meant a high priestess who resided in a sanctuary located on Mt. Parnassus, and gave predictions and advice to both statesmen and citizens.

2 Mapping Reducibility

We now consider a special kind of Turing reduction from problem A to problem B for which given problem instance x of A, i) exactly one query is allowed to the B-oracle, and ii) the answer provided by the B-oracle equals the solution to x, Because of this we may assume that there exists a computable function $f : A \to B$ that represents the query to the B-oracle. In other words, given instance x of A, $f(x)$ represents the single query to B whose answer/solution is the answer/solution for x.

Definition 2.1. Problem A is **mapping reducible** to problem B, written $A \leq_m B$, iff there exists a computable function $f : A \to B$ for which the solution to problem instance x of A is equal to the solution to problem instance $f(x)$ of B.

Notes:

1. Similar to Turing reducibility, if we insist that the algorithm for computing f requires at most a polynomial number of steps with respect to the size of an instance x, then we say A is **polynomial-time mapping reducible** to B and write $A \leq_m^p B$.
2. The term *map* is a synonym for *function*.
3. A special case of a mapping reduction occurs when A and B are decision problems in this case f has the property that x is a positive instance of A iff $f(x)$ is a positive instance of B. In other words, a positive (respectively, negative) instance of A must map to a positive (respectively, negative) instance of B.

3 Basic Examples

Example 3.1. We begin with a toy example by considering the two decision problems `Even` and `Odd` where an instance of either problem is an integer n. Moreover, for `Even` the problem is to decide if n is even. Problem `Odd` is similarly defined. Then we have `Even` $\leq_m$ `Odd` via function $f(n) = n + 1$. This is true since n is even if and only if $f(n) = n + 1$ is odd. Thus, the answer to n equals the answer to $f(n)$. Notice that f also provides a reduction from `Odd` to `Even`.

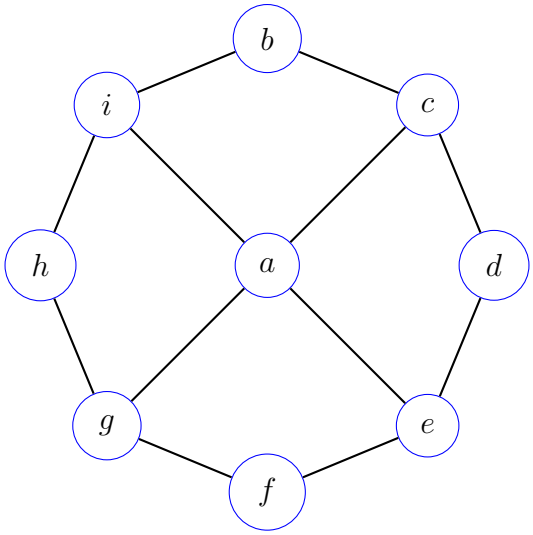


Figure 1: $I = \{a, b, d, f, h\}$ is an independent set for the above graph

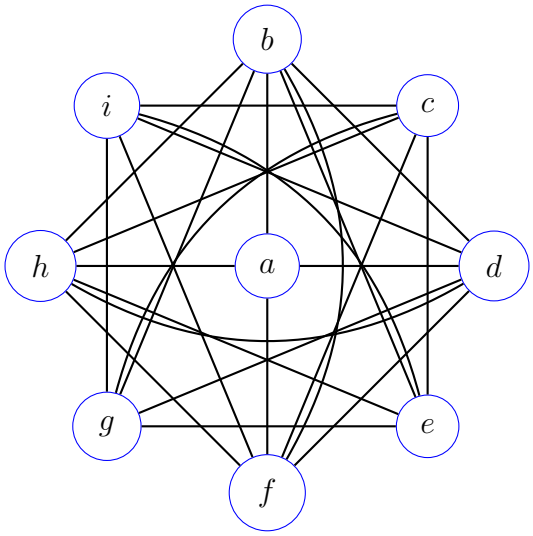


Figure 2: The complement graph $\overline{G}$ for the graph G shown in Figure 1

Example 3.2. Recall the `Max Clique` optimization problem involves determing the size of the largest clique in some graph. A related problem is the `Max Independent Set (MIS)` optimization problem for which a problem instance is a simple graph $G = (V, E)$ and the problem is to determine the size of the largest subset $I \subseteq V$ of vertices such that, for every $u \in I$ and $v \in I$, $(u, v) \notin E$. In other words, every pair of vertices that both belong to I must be non-adjacent. Figure 1 shows a graph whose largest independent set is $I = \{a, b, d, f, h\}$.

We now provide a map reduction from `MIS` to `Max Clique`. Given a simple graph $G = (V, E)$, our reduction makes use of the **complement** of G, denoted $\overline{G}$, which is defined as $\overline{G} = (V, \overline{E})$, where, for any two vertices $u, v \in V$, $(u, v) \in \overline{E}$ iff $(u, v) \notin E$. In other words, G and its complement $\overline{G}$ have the same vertex set, but the edges of $\overline{G}$ are exactly those edges that are *not* edges of G, and vice versa. For example, the Graph is Figure 2 shows the complement of the graph G in Figure 1.

Notice that the independent set $I = \{a, b, d, f, h\}$ for G now represents a 5-clique in $\overline{G}$. In fact, this is true for *any* independent set I for G: I is an independent set for G iff I is a clique for $\overline{G}$. Thus, we may provide a map reduction $f : \texttt{MIS} \to \texttt{Max Clique}$ that is defined by $f(G) = \overline{G}$. Indeed k is the size of the largest independent set of G iff it is the size of the largest clique of $\overline{G}$. □

In the previous example, it is important to note that the same map $f : \texttt{Max Clique} \to \texttt{MIS}$ may be used to reduce `Max Clique` to `MIS`. This because both problems have the same domain (simple graphs) and C is a maximum clique for G iff C is a maximum independent for $f(G) = \overline{G}$.

4 Embeddings

Definition 4.1. An **embedding reduction** is a kind of mapping reduction for which a problem A is map reduced to problem B, where A is a special case of B.

As an example, consider a vehicle that has an air-conditioning system that includes the ability to cool and heat different parts of the vehicle, as well as control the flow of air entering and leaving the vehicle. Recall our friend Sam from the Turing Reducibility lecture. Sam and his friends drove to the Mohave desert where they encountered extreme heat, lots of wind, and some very dusty roads. While driving, they reduced the problem of staying cool and breathing clean air to setting the vehicle's air-conditioner to "cool" and allowing no outside air to enter the vehicle. In other words, the general air-conditioning system served the special purpose of cooling and maintaining air cleanliness.

Definition 4.2. The `Subset Sum (SS)` decision problem is a pair (S, t), where S is a set of nonnegative integers, and t is a nonnegative integer. The problem is to decide if there is a subset $A \subseteq S$ whose members sum to t, i.e., a subset A for which

$$\sum_{a \in A} a = t.$$

Example 4.3. `Subset Sum` instance $(S = \{3, 7, 13, 19, 22, 26, 35, 38, 41\}, t = 102)$ is a positive instance of `SS` since $A = \{3, 7, 13, 38, 41\} \subseteq S$ and

$$3 + 7 + 13 + 38 + 41 = 102.$$

Definition 4.4. An instance of decision problem `Set Partition (SP)` consists of a set S of positive integers, and the problem is to decide if there are subsets $A, B \subseteq S$ for which

1. $A \cap B = \emptyset$,
2. $A \cup B = S$, and
3. $$\sum_{a \in A} a = \sum_{b \in B} b.$$

In other words, the members of A must sum to the same value as the members of B.

Example 4.5. Show that `Set Partition` instance

$$S = \{3, 14, 19, 26, 35, 37, 43, 49, 52\}$$

is a positive instance of `SP`.

A few moments of thought should convince you that `SP` is a special case of `SS`. Indeed given instance S of `SP`, if we let M denote the sum of all members of S, then we essentially are seeking a subset A whose members sum to $M/2$ since, if such A can be found, then by setting $B = S - A$, we have all three conditions met (check this!). Therefore, we may reduce `SP` to `SS` via the map

$$f(S) = (S, t = M/2), \tag{1}$$

where M is defined as above.

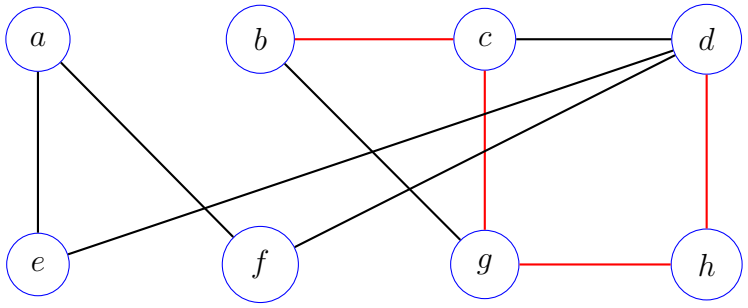
Just as Sam and his friends don't care about the heating features of the air-conditioning, when solving an instance of `Set Partition` via `Subset Sum`, we don't care that `Subset Sum` can solve a wider variety of problems involving t values that may not have any relationship with S.

Example 4.6. Given the instance $S = \{4, 8, 17, 25, 34, 46, 53, 59\}$ of `Set Partition`, provide the instance of `SS` to which it reduces via the mapping defined in equation 1.

Solution.

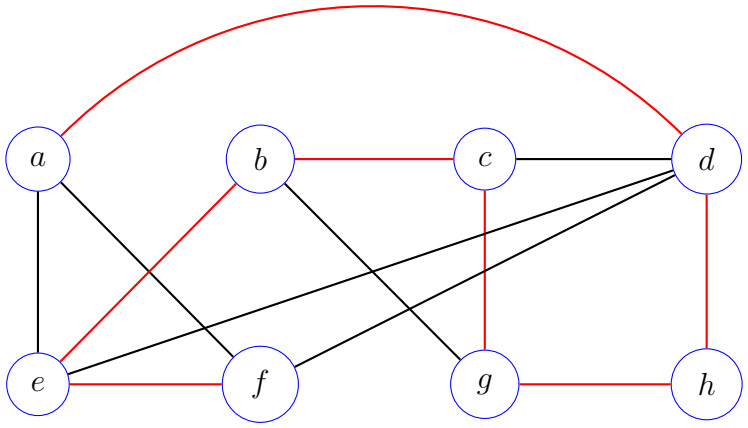
Definition 4.7. An instance of the `LPath` decision problem is a pair (G, k), where $G = (V, E)$ is a simple graph, and $k \geq 0$ is a nonnegative integer. The problem is to decide if G has a **simple path** of length k, i.e. a path that traverses k edges and visits exactly $k + 1$ different vertices.

Example 4.8. The following graph, along with $k = 4$, shows a positive instance of LPath.



Definition 4.9. An instance of the `Hamilton Path (HP)` decision problem consists of a simple graph $G = (V, E)$ and the problem is to decide if G has a **Hamilton path**, namely a path that visits every vertex in G exactly once.

Example 4.10. The following graph represents a positive instance of `HP`, with the Hamilton Path shown in red.



Once again, it is hopefully evident that `HP` is a special case of `LPath`. To see this, note that a simple graph has a Hamilton path iff it has a simple path of length $n - 1 = |V| - 1$. Therefore, we may reduce `HP` to `LPath` via the map

$$f(G) = (G, |V| - 1).$$

□

5 Contractions

As the examples in the previous section suggest, devising an embedding reduction from problem A only requires knowledge of a problem B that is more general than A and includes A as a special case. On the other hand, a **contraction reduction** is a mapping reduction for which problem B is the special case of problem A. Defining a contraction usually involves some insight and imagination. Moreover, a contraction reduction demonstrates actual computing progress in the sense that a body A of problem instances gets effectively reduced to a smaller set B which may improve the chances of efficiently solving A through the lens of B.

Example 5.1. Contractions are quite common in everyday computing. As an example, consider a legacy compiler $\mathcal{C}$ for some programming language L. Over the years L gets extended with the addition of new programming constructs that improve the ease of programming in L. We'll call this extended version Turbo-L. Rather than write a new compiler for Turbo-L, we instead provide a contraction that is capable of translating any Turbo-L program to an L program, followed by compiling the L-code with the legacy compiler. □

Let's return to the `Subset Sum (SS)` and `Set Partition (SP)` problems defined in Section 4. There we showed a natural embedding from `SP` to `SS`. Now we show a contraction from `SS` to `SP`. To see how the contraction works, consider the set

$$S = \{7, 12, 15, 23, 37, 42, 48\}.$$

The sum of its members is $M = 184$. Therefore, if (S, t) is an instance of `SS` with $t = 184/2 = 92$, then, by dropping t, this instance naturally maps to `SP`:

$$(S, t) \rightarrow S,$$

and (S, t) is positive for `SS` iff S is positive for `SP`.

What if t is less than half of M? Then we may not simply drop t, but will also have to modify S. But how? As an example, suppose that $t = 64 < 92$. Then (S, t) is a positive instance of `SS` via $A = \{12, 15, 37\}$. Notice also that the members of the complement $B = S - A$ sums to 120. Therefore, by adding to S the extra number $J = 56$, we see that the members of

$$A \cup \{56\} = \{12, 15, 37, 56\},$$

also sum to 120 and thus $S' = S \cup \{56\}$ is a positive instance of `SP`.

Let's generalize the example from the last paragraph. Assume that $t < M/2$. Then we must add the number J to S for which

$$t + J = M - t \Rightarrow J = M - 2t.$$

Now suppose (S, t) is positive for `SS` via $A \subseteq S$. Let $A' = A + \{M - 2t\}$ and $B' = S - A$. Then we have

1. $A' \cap B' = \emptyset$,
2. $A' \cup B' = S' = S + \{M - 2t\}$, and
3. the members of A' sum to $t + (M - 2t) = M - t$ while the members of B' also sum to $M - t$.

Therefore, $S' = S \cup \{M - 2t\}$ is a positive instance of `SP`.

We now show that the converse is also true: if $S' = S \cup \{M - 2t\}$ is positive for `SP`, then there are subsets A' and B' of S' for which

1. $A' \cap B' = \emptyset$,
2. $A' \cup B' = S'$,
3. $(M - 2t) \in A'$, and
4. the members of $A = A' - \{M - 2t\}$ sum to
$$1/2(M + (M - 2t)) - (M - 2t) = (M - t) - (M - 2t) = t.$$

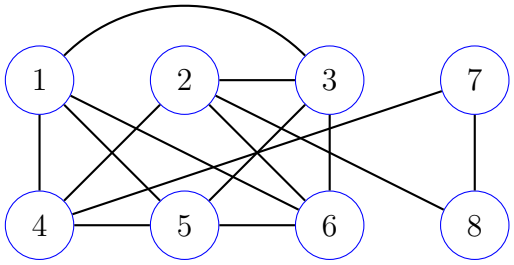
And since the members of $A = A' - \{M - 2t\}$ are all members of S, we see that (S, t) is positive for `SS`.

Example 5.2. Derive the formula for J in case $t > M/2$.

Example 5.3. Use the above reduction to map the following instances of `Subset Sum` to `Set Partition`.

1. $(\{7, 11, 23, 25, 37, 39, 49, 73\}, t = 51)$
2. $(\{7, 11, 23, 25, 37, 39, 49, 73\}, t = 132)$
3. $(\{7, 11, 23, 25, 37, 39, 49, 73\}, t = 200)$

Example 5.4. The `Vertex Cover (VC)` decision problem is the problem of deciding if a simple graph $G = (V, E)$ has a vertex cover of size $k \geq 0$, for some integer k. In other words does G have a subset C of k vertices for which every edge $e \in E$ is incident with at least one vertex in C? Show that $(G, k = 5)$ is a positive instance of `VC`, where G is shown below.



Solution.

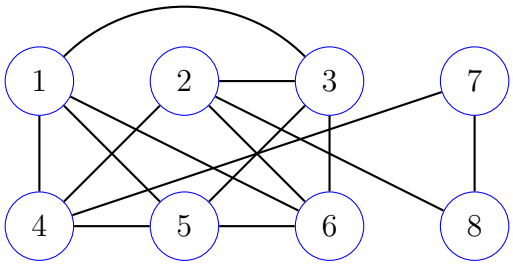


Figure 3: Graph G_1 for Example 5.5.

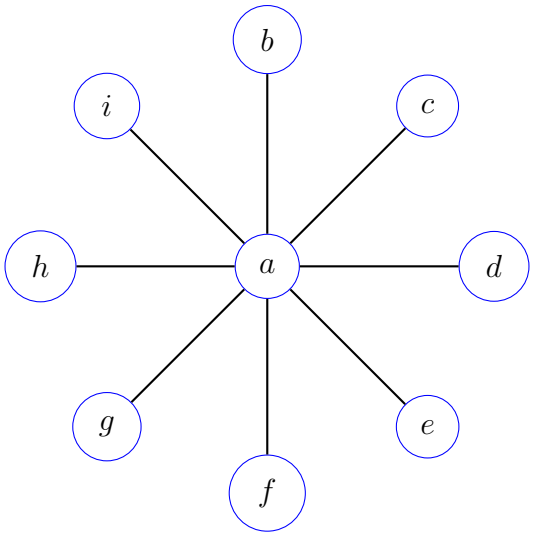


Figure 4: Graph G_2 for Example 5.5.

Example 5.5. Decision problem `Half Vertex Cover (HVC)` is similar to `Vertex Cover (VC)`, except now an instance consists of a graph $G = (V, E)$, and the problem is to decide if G has a vertex cover whose size equals $|V|/2$. Provide a contraction reduction from `VC` to `HVC`. Apply the reduction to the `VC` instances $(G_1, k = 5)$ and $(G_2, k = 1)$, where G_1 and G_2 are shown in the Figures 3 and 4.

6 The 3SAT Logic Problem

In this Section we introduce the 3SAT logic problem which will which represents one of the more important problems in all of Computer Science.

6.1 Boolean Variable Assignments

Before introducing the 3SAT decision problem, we need to understand the concept of a Boolean variable assignment.

Boolean Variable A variable is said to be **Boolean** iff its domain equal $\{0, 1\}$. We use lowercase letters, such as $x, y, z, x_1, x_2, \ldots$, etc., to denote a Boolean variable.

Assignment An **assignment** over a Boolean-variable set V is a function $\alpha : V \to \{0, 1\}$ that assigns to each variable $x \in V$ a value in $\{0, 1\}$. We may represent α using function notation, or as a labeled tuple.

Example: for the assignment α that assigns 1 to both x_1 and x_2, and 0 to x_3, we may use function notation and write $\alpha(x_1) = 1$, $\alpha(x_2) = 1$, and $\alpha(x_3) = 0$, or we may use tuple notation and write

$$\alpha = (x_1 = 1, x_2 = 1, x_3 = 0),$$

or

$$\alpha = (1, 1, 0),$$

if the associated variables are understood.

Variable Negation If x is a variable, then $\overline{x}$ is called its **negation**.

Example: Suppose assignment α satisfies $\alpha(x_1) = 0$. Then (extending α to include negation inputs) $\alpha(\overline{x}_1) = 1$.

Literal A **literal** is either a variable or the negation of a variable .

Example: x_1, x_3, $\overline{x}_3$, are $\overline{x}_5$ all examples of literals.

Consistent A set R of literals is called **consistent** iff no variable and its negation are both in R. Otherwise, R is said to be **inconsistent**.

Example: $\{x_1, \overline{x}_2, x_4, \overline{x}_7, \overline{x}_9\}$ is a consistent set, but $\{x_1, \overline{x}_2, x_4, \overline{x}_7, x_7\}$ is an inconsistent set.

Induced Assignment If $R = \{l_1, \ldots, l_n\}$ is a consistent set of literals, then α_R is called the **(partial) assignment induced by R** and is defined by $\alpha(l_i) = 1$ for all $l_i \in R$.

Example: the assignment induced by $R = \{x_1, \overline{x}_2, x_4, \overline{x}_7, \overline{x}_9\}$ is

$$\alpha = (x_1 = 1, x_2 = 0, x_4 = 1, x_7 = 0, x_9 = 0).$$

Definition 6.1. A **ternary disjunctive clause** is a Boolean formula of the form

$$l_1 \vee l_2 \vee l_3,$$

where l_1, l_2, and, l_3 are literals. The clause evaluates to 1 in case at least one of l_1, l_2, or l_3 is assigned 1. In this case we say the clause is **satisfied**. Otherwise it is **unsatisfied**.

Definition 6.2. An instance of the `3SAT` decision problem consists of a set $\mathcal{C}$ of ternary disjunctive clauses. The problem is to decide if there is an assignment α over the variables in $\mathcal{C}$, such that every clause $(l_1 \vee l_2 \vee l_3)$ in $\mathcal{C}$ evaluates to 1 under α. If such an assignment α exists, then it is said to be a **satisfying assignment** and we say $\mathcal{C}$ is **satisfiable**. Otherwise, $\mathcal{C}$ is said to be **unsatisfiable**. Finally, the `3SAT` decision problem is the problem of deciding whether a set $\mathcal{C}$ of ternary clauses is sastisfiable.

Simplified clause notation. In what follows, we often simplify the clause notation by writing each clause $(l_1 \vee l_2 \vee l_3)$ as (l_1, l_2, l_3).

Example 6.3. Provide a satisfying assignment for

$$\mathcal{C} = \{(x_1, x_2, x_3), (\overline{x}_2, x_3, \overline{x}_4), (x_1, x_2, \overline{x}_4), (\overline{x}_1, \overline{x}_3, \overline{x}_4), (\overline{x}_1, x_2, x_4), (\overline{x}_2, x_3, x_4), (x_1, \overline{x}_3, x_4),$$

$$(\overline{x}_2, \overline{x}_3, x_4), (\overline{x}_2, \overline{x}_3, \overline{x}_4)\}.$$

7 Interdomain reductions

In this section we look at **interdomain** mapping reductions that reduce a problem from one domain to a problem in a different domain. Some of the different mathematical and computer science domains include the following. include

Mathematics logic, graph theory, algegra and number theory, numerical optimization, geometry

Computer Science machine learning, network design and analysis, data storage and retrieval, cryptography/security, operating systems, automata and languages, programming languages and program optimization.

Of course, as is witnessed by interdomain reductions, different domains are often related in several ways, and thus there is some subjectivity regarding the classifications of problems. Nevertheless, the above mentioned domains are considered separate and vast areas of research. As we'll see in the following examples, interdomain reductions are often the more surprising and clever of all reductions.

The reductions we study in this section both reduce from the `3SAT` logic problem. Because of the ability to reduce `3SAT` to other problem domains, `3SAT` plays a crucial role in the study of NP-completness which we cover in the next lecture.

The following theorem uses refers to `Clique`, the decsion-problem version of `Max Clique`. In this case, an instance of the `Clique` is a simple graph $G = (V, E)$ and an integer $k \geq 0$. The problem is to decide if there is a subset $C \subseteq V$ of k vertices that are pairwise adjacent.

Theorem 7.1. `3SAT` $\leq_m^p$ `Clique`.

Proof. Let $\mathcal{C}$ be a collection of m clauses, where clause c_i, $1 \leq i \leq m$, has the form $c_i = l_{i1} \vee l_{i2} \vee l_{i3}$. We now define a mapping $f(\mathcal{C}) = (G, k = m)$ for which G has an m-clique if and only if $\mathcal{C}$ is satisfiable. $G = (V, E)$ is defined as follows. V consists of $3m$ vertices, one for each literal l_{ij}, $1 \leq i \leq m$ and $1 \leq j \leq 3$. Then $(l_{ij}, l_{rs}) \in E$ iff i) $i \neq r$ and ii) l_{ij} is not the negation of l_{rs} (i.e. the two literals are logically consistent).

First assume that $\mathcal{C}$ is satisfiable. Given a satisfying assignment α for $\mathcal{C}$, let l_{ij_i}, $1 \leq i \leq m$, denote a literal from clause i that is satisfied by α, i.e. $\alpha(l_{ij_i}) = 1$. Then, since each pair of these literals is consistent, $(l_{ij_i}, l_{rj_r}) \in E$ for all $i \neq r$. In other words,

$$C = \{l_{1j_1}, l_{2j_2}, \ldots, l_{mj_m}\}$$

is an m-clique for G.

Conversely, assume G has an m-clique. Then by the way G is defined the clique must have the form

$$C = \{l_{1j_1}, l_{2j_2}, \ldots, l_{mj_m}\}$$

where l_{ij_i} is a literal in c_i. This is true since no two literal vertices from the same clause can be adjacent and so each literal vertex in C must come from a different clause. Moreover, by the definition of G, C is a consistent set of literals. Hence the assignment a_C induced by C satisifes every clause of $\mathcal{C}$, since $\alpha_C(l_{ij_i}) = 1$ satisfies c_i, for each $i = 1, \ldots, m$. Therefore, $\mathcal{C}$ is satisfiable.

Finally, we must show is that $f(\mathcal{C})$ may be computed via an algorithm whose running time is polynomial in m and n. But this can be done via two nested `for`-loops that iterate through each pair of clauses i and r, $i \neq r$, and identify all consistent pairs of literals (l_{ij_i}, l_{rj_r}). This require $\mathrm{O}(m^2)$ steps. □

Example 7.2. Show the reduction provided in the proof of Theorem 7.1 for input instance

$$\mathcal{C} = \{(x_1, x_1, x_2), (\overline{x}_1, \overline{x}_2, \overline{x}_2), (\overline{x}_1, x_2, x_2)\}.$$

Theorem 7.3. 3SAT $\leq_m^p$ Subset Sum.

Let $\mathcal{C}$ be a collection of m ternary clauses over n variables. The following table provides the reduction $f(\mathcal{C}) = (S, t)$. The table rows correspond to the $(n+m)$-digit integers comprising set S, while t is the integer at the bottom whose first n digits are 1's and whose final m digits are 3's.

	1	2	3	4	$\cdots$	n	c_1	c_2	$\cdots$	c_m
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
z_2		1	0	0	$\cdots$	0	0	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$						$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_n						1	0	0	$\cdots$	0
z_n						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$										$\vdots$
g_m									$\cdots$	1
h_m									$\cdots$	1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

Number y_i correpsonds to literal x_i, while z_i corresponds to literal $\overline{x}_i$. Thus, since the first n digits of t are 1's, we see that, to construct a subset A whose members sum to t, it must either contain y_i or z_i, but not both. Also, the final m digits of y_i (respectively, z_i) indicate which clauses c_i are satisfied by x_i (respectively $\overline{x}_i$).

Now suppose $\mathcal{C}$ is satisfiable via some assignment α. Then the subset A needed to sum to t includes the following numbers. For $1 \leq i \leq n$, if $\alpha(x_i) = 1$, then add y_i to A; otherwise add z_i to A. For $1 \leq j \leq m$, to determine if g_j and/or h_j should be added to A, consider clause $c_j = \{l_{j1}, l_{j2}, l_{j3}\}$ and the sum

$$\sigma_j = \alpha(l_{j1}) + \alpha(l_{j2}) + \alpha(l_{j3}).$$

Since α satisfies $\mathcal{C}$, we must have $\sigma_j \geq 1$.

Case 1: $\sigma_j = 1$. In this case add both g_j and h_j for the c_j-column to sum to 3.

Case 2: $\sigma_j = 2$. In this case add only g_j for the c_j-column to sum to 3.

Case 3: $\sigma_j = 3$. In this case neither g_j nor h_j need to be added since the c_j-column already sums to 3.

. Therefore, it is always possible to find a subset A whose members sum to t.

Conversely, suppose there is a subset $A \subseteq S$ whose members sum to t. Then for each $i = 1, \ldots, n$, either $y_i \in A$ or $z_i \in A$, but not both. This is true since t's first n digits are 1's. Let α be an assignment over the variables of $\mathcal{C}$ such that, for each $i = 1, \ldots, n$ $\alpha(x_i) = 1$ iff $y_i \in A$. Now consider clause c_j, $j = 1, \ldots, m$. We know that the digits of the members of A in the c_j-column sum to 3. Thus, one of the members must either be y_k or z_k for some $k = 1, \ldots, n$. In other words, either $y_k \in A$, $x_k \in c_j$, and $\alpha(x_k) = 1$ or $z_k \in A$, $\overline{x}_k \in c_j$ and $\alpha(\overline{x}_k) = 1$. In either case, we see that α satisfies c_j. Therefore, since $j = 1, \ldots, n$ was arbitrary, α satisfies $\mathcal{C}$.

Finally, notice that each of the $2m + 2n$ integers in S can be constructed in $\mathrm{O}(m + n)$ steps, and so $f(\mathcal{C}) = (S, t)$ can be computed in $\mathrm{O}(m^2 + n^2)$ steps. □

Example 7.4. Show the reduction given in Theorem 7.3 using input instance

$$\mathcal{C} = \{c_1 = (x_1, x_2, x_3), c_2 = (\overline{x}_1, \overline{x}_2, \overline{x}_3), c_3 = (\overline{x}_1, x_2, x_3), c_4 = (x_1, \overline{x}_2, \overline{x}_3)\}.$$

Solution.

	1	2	3	c_1	c_2	c_3	c_4
y_1	1	0	0	1	0	0	1
z_1	1	0	0	0	1	1	0
y_2		1	0	1	0	1	0
z_2		1	0	0	1	0	1
y_3			1	1	0	1	0
z_3			1	0	1	0	1
g_1				1	0	0	0
h_1				1	0	0	0
g_2					1	0	0
h_2					1	0	0
g_3						1	0
h_3						1	0
g_4						0	1
h_4						0	1
t	1	1	1	3	3	3	3

8 Proving undecidability with a mapping reduction via the SMN Theorem

Theorem 8.1. Consider two decision problems A and B where A is undecidable and $A \leq_m B$. Then B is also undecidable.

Proof. Suppose, by way of contradiction, B is decidable via some URM program $\mathcal{B}$. Let $f : A \to B$ denote the URM-computable function that witnesses $A \leq_m B$. Then the following is an informal algorithm for deciding A. On input x, compute $f(x)$. Then place the result $f(x)$ as input to $\mathcal{B}$ and accept x iff $\mathcal{B}$ accepts $f(x)$, since x and $f(x)$ must both be either positive or negative instances of their respective problems A and B. It follows that A is decidable, a contradiction. Therefore, B must be undecidable. □

We now show how the SMN Theorem can be used to define the $f(x)$ described in the above proof. Given undecidable problem A, design a function $g(x, y)$, so that

1. If x_0 is a positive instance of A, then the function of y $g(x_0, y)$, obtained by substituting x_0 for x, has an index (i.e. Gödel number) that is associated with a positive instance of B.
2. If x_0 is a negative instance of A, then the function of y $g(x_0, y)$, obtained by substituting x_0 for x, has an index (i.e. Gödel number) that is associated with a negative instance of B.

Now the SMN Theorem guarantees a total computable function $f(x)$ for which

$$\phi_{f(x)}(y) = g(x, y).$$

Therefore, if x is a positive (respectively, negative) instance of A, then index $f(x)$ for the function $g(x, y)$ is a positive (respectively, negative) instance of B. In other words $A \leq_m B$ via reducing function $f(x)$.

Example 8.2. An instance of the decision problem `Zero` is a Gödel number x, and the problem is to decide if function ϕ_x equals the zero function, i.e. the function that returns 0 on every input. Show that `Self Accepting` $\leq_m$ `Zero`.

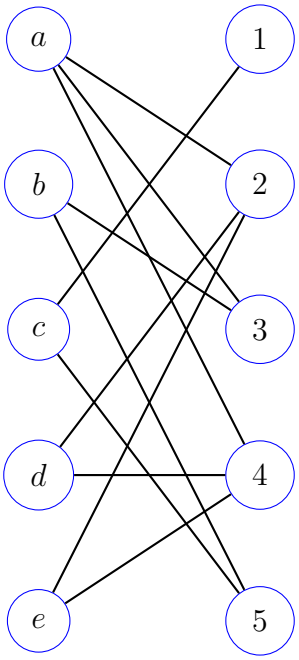


Figure 5: An example of a bipartite graph

9 Finding maximum matchings in bipartite graphs

The mapping reducibilities provided in Examples 3.1 and 3.2 involved pairs of problems that were in some sense two sides of the same coin. Our next example involves a more subtle relationship between two seemingly unrelated graph problems, namely the `Max Flow` problem and the `Maximum Bipartite Matching (MBM)` problem. We've already encountered the former Section 6 of the Turing Reducibility lecture, and now describe the latter before providing a map reduction from `MBM` to `Max Flow`.

A **bipartite** graph $G = (V_1, V_2, E)$ consists of two nonempty disjoing sets of vertices V_1 and V_2 and a set of edges for which each edge is incident with one vertex in V_1 and one vertex in V_2. Figure 5 shows an example of a bipartite graph, with the $V_1 = \{a, b, c, d, e\}$ and $V_2 = \{1, 2, 3, 4, 5\}$.

A **matching** M in the graph is a subset $M \subseteq E$ of edges of G, no two of which share a common vertex. Figure 6 shows a matchiing for the graph in Figure 5.

The following definitions prove useful when discussing matchings in a graph. Let M be a matching.

Maximum Matching M is a called a **maximum** matching iff for any other matching M' of G, $|M'| \leq |M|$.

Maximal Matching M is called **maximal** iff it is not contained in any larger matching. In other words, one cannot increase the size of M simply by adding another edge to M.

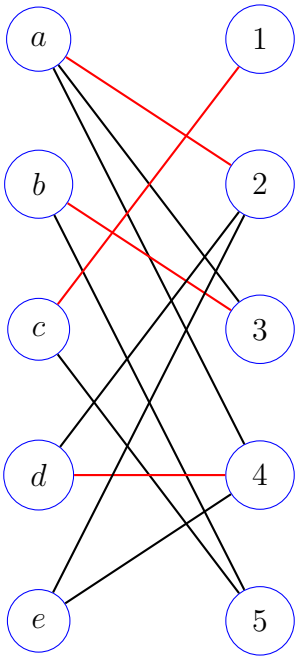


Figure 6: The red edges form a maximal matching for the bipartite graph.

Matched and Free Edges The edges of M are called **matched** edges while the edges in $E - M$ are called **free** edges.

Exposed and Covered Vertices Any vertex that is not incident with an edge in M is said to be **exposed** by M. otherwise it is **covered** by M.

Increment Matching M^* is called an **increment** of M iff $|M^*| = |M| + 1$

Notice that the matching M in Figure 6 is maximal, since no edge can be added to M to increase its size. Indeed, the only exposed vertices are e and 5, but they are not adjacent. However, M is not a maximum matching. A maximum matching is shown in Figure 7. This matching is an increment of M. Finally, `MBM` is the problem of finding the size of a maximum matching in a bipartite graph.

We now describe a map reduction from `MBM` to `Max Flow`. Let $G = (V_1, V_2, E)$ be a bipartite graph. Then

$$f(G) = G' = (V', E', c, s, t),$$

where the directed network G' is defined as follows.

Vertices $V' = V_1 \cup V_2 \cup \{s, t\}$, where s is the source vertex and t is the destination vertex

Edges i) $(s, u) \in E'$ for each vertex $u \in V_1$, ii) $(v, t) \in E'$ for each vertex $v \in V_2$, and iii) if $e = (u, v) \in E$ is an edge of G, with $u \in V_1$ and $v \in V_2$, then $e = (u, v) \in E'$ is a directed edge of G'. Note: such an edge are called an **original edge** because it also exists in G.

Capacity all edges are assigned unit (i.e. 1) capacity

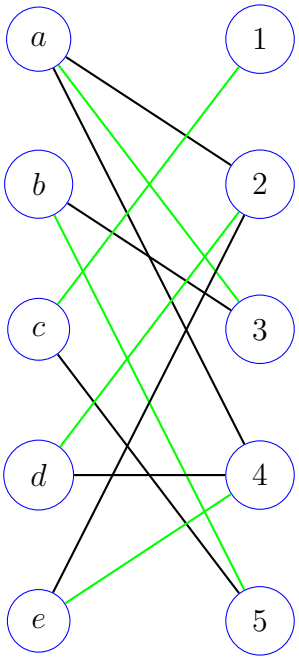


Figure 7: The green edges form a maximum matching for the bipartite graph.

Figure 8 shows $f(G)$ for the instance G of `MBM` shown in Figure 5.

Theorem 9.1. The above-described mapping from `MBM` to `Max Flow` is a mapping reduction, i.e. G has a matching of size k iff G' has a flow of size k.

Proof. Suppose M is a matching for G, with $|M| = k$. Then there exists a flow f_M on G' with the following property. For each edge $e = (u, v) \in M$,

$$f_M(s, u) = f_M(e) = f_M(v, t) = 1,$$

and $f_M(e) = 0$ for all other edges $e \in E'$. Figure 9 shows f_M for the network $G' = f(G)$, where G and M are the respective bipartite graph and matching shown in Figure 6.

To show that f_M is a flow, first note that $f_M(e) \leq 1$ and thus f_M never exceeds the unit capacity limit of e. We now show that each vertex $u \in V_1 \cup V_2$ preserves flow. Without loss of generality, assume $u \in V_1$. Case 1: u is exposed by M. Then there is no edge $e \in M$ that is incident with u. Then, by definition of f_M, there is zero flow both entering and leaving u. Case 2: u is covered by M. Then $f_M(s, u) = 1$ and there is unit flow entering u. Also, there is a unique edge $e \in M$ for which $e = (u, v)$, for some $v \in V_2$. Thus, $f_M(e) = 1$ and there is unit flow leaving u. Therefore, u conserves flow. Finally, notice that $s(f_M) = |M| = k$, the size of M.

Now assume $G' = f(G)$ has a flow f of size k, where k is a nonnegative integer. Based on the `Max-Flow` algorithm, we may assume that $f(e)$ is either 0 or 1, for every $e \in E'$. Let M denote the set of original edges e of G' for which $f(e) = 1$. Then we claim that M is a matching for G and that $|M| = k$. To see this, since $s(f) = k$, there are exactly k edges of the form (s, u) for which $f(s, u) = 1$. Then for each u for which $f(s, u) = 1$ there is a unique vertex $v_u \in V_2$ for which $f(u, v_u) = 1$. This

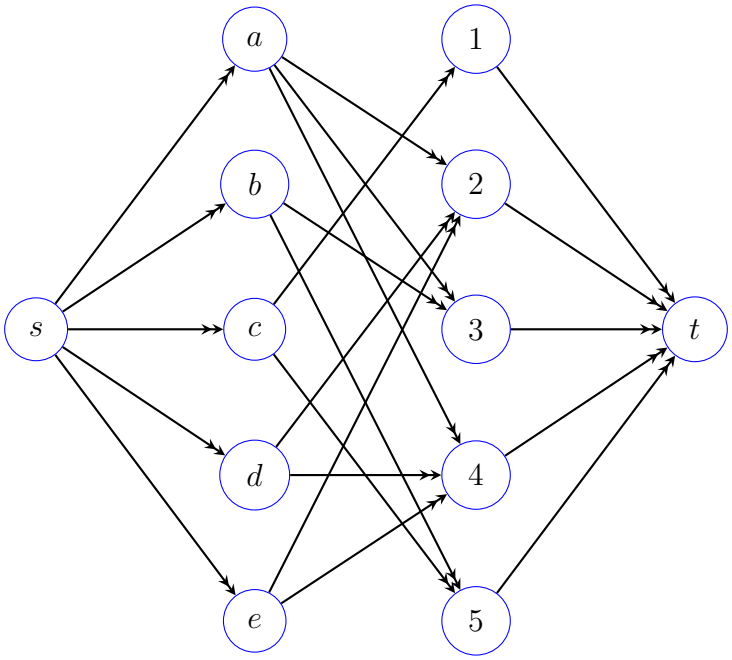


Figure 8: The directed network $G' = f(G)$, where G is the graph in Figure 5.

is because u must conserve flow and there is one unit of flow entering u. It follows then that

$$M = \{(u, v_u) | f(s, u) = 1$$

is a matching and that $|M| = k$. □

When reducing `MBM` to `Max Flow` using the mapping defined above, each residual network G'_f that results from a flow f through G' can be easily drawn using the following rule: for each $e \in E'$, if $f(e) = 1$, then e is oriented backwards in G'_f. Otherwise it remains forward oriented. Furthermore, what we earlier referred to as an augmenting path is now called an **alternating path** because, upon leaving s, it has the property of first reaching an exposed vertex in V_1, followed by alternating between forward and backward edges until it reaches an exposed vertex of V_2, and finally reaches t. Figure 10 shows the residual network G'_{f_M} for the network G' and flow f_M showed in Figure 9. Also, Figure 11 shows an alternating path (in green) for G'_{f_M}.

Finally, there is a nice way to characterize the increment matching M' that one obtains from alternating path P. Namely, $M' = M \oplus P$, where the (edge) set operation $M \oplus P$ is called the **symmetric difference** between M and P and consists of all edges that are either in M but not P, or in P but not M (here, we neither include the edge that leaves s, nor the one that enters t).

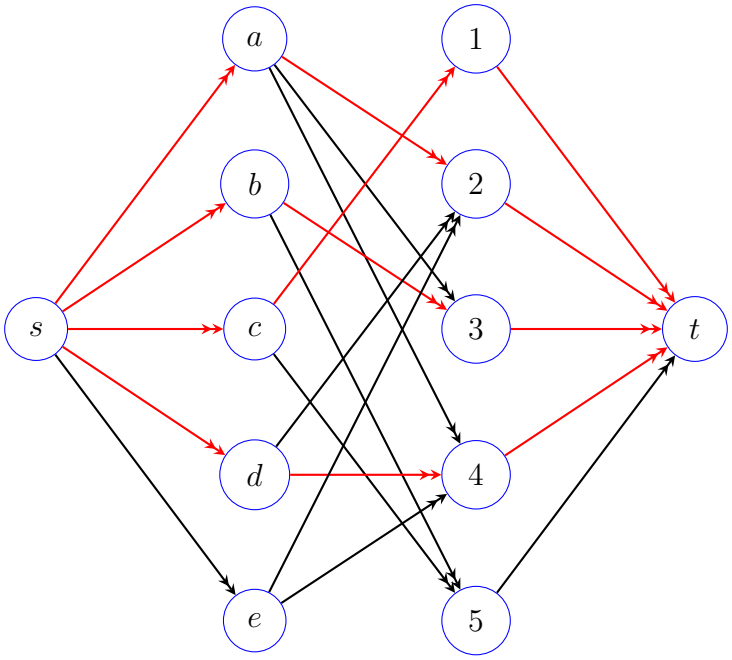


Figure 9: The network $G' = f(G)$, with flow f_M (in red) associated with matching M from Figure 6.

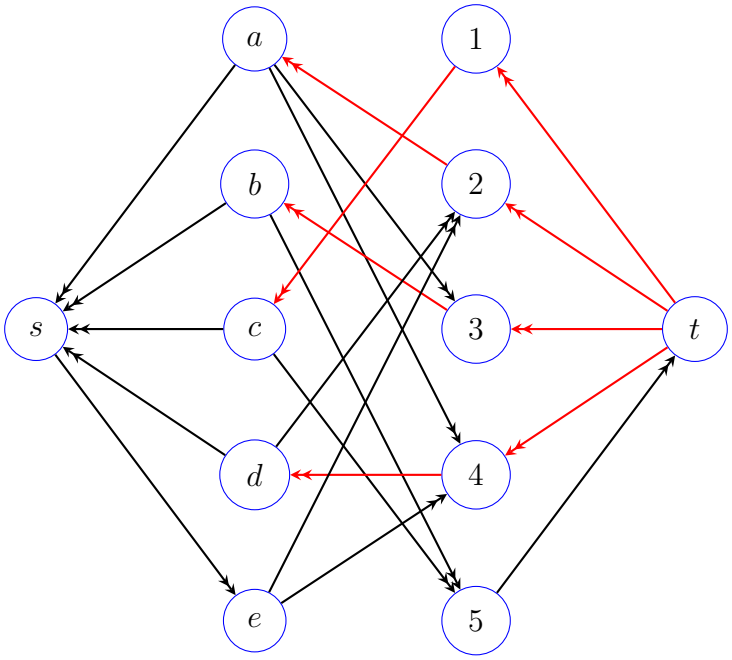


Figure 10: Residual network G'_{f_M} for graph G' and flow f_M from Figure 9.

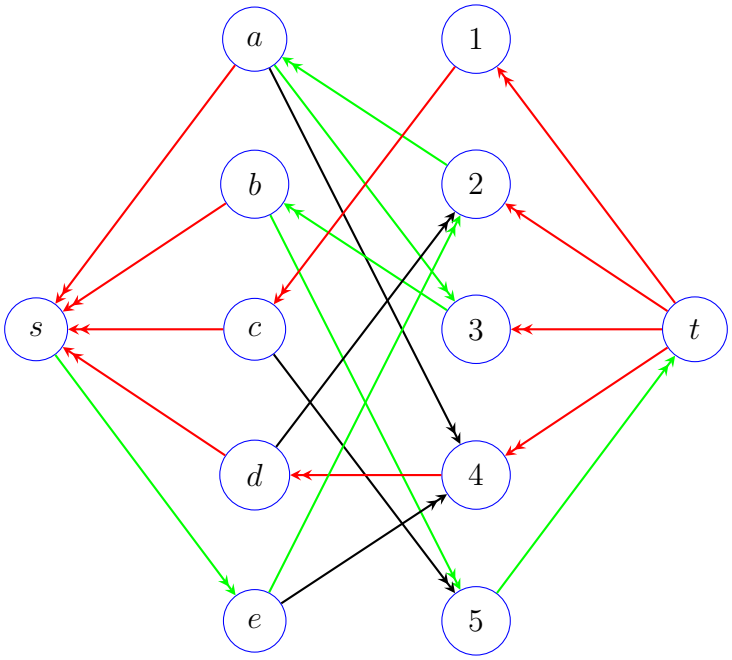


Figure 11: Alternating path (in green) for residual network G'_{f_M} of Figure 10.

Maximum Matching Algorithm

Input: bipartite graph G.

Output: a maximum matching for G.

Compute $f(G) = G' = (V', E', c, s, t)$.

Initialize matching M: $M \leftarrow \emptyset$.

While there is a vertex of G that is exposed by M

If G'_{f_M} has no alternating path, then return M.
Let P be an alternating path of G'_{f_M}.
$M \leftarrow P \oplus M$.

Return M.

The **Perfect Bipartite Matching (PBM)** decision problem is the problem of deciding if a bipartite graph G has a **perfect matching**, i.e. one that covers every vertex of G. For a bipartite graph to have such a matching, it must be the case that $|V_1| = |V_2|$. Moreover, `PBM` mapping reducible to the decsion-problem version of `Max Flow`, where a problem instance is now a network G and an integer k, and the problem is to decide if G admits a flow of size k.

10 Some Consequences of Reducibility

If we have a reduction (Turing or mapping) from problem A to problem B, what can we infer about either of the problems? Two things we may be able to learn involve the notions of solvability and complexity.

We say that, e.g., problem A is **solvable** iff there is an algorithm that takes as input any instance x of A and computes the solution to that instance. If no such algorithm exists, then we say that A is **unsolvable**. In a later Chapter give examples of several unsolvable problems and techniques for proving their unsolvability.

Now, assuming A is solvable, the complexity of solving A refers to determining lower bounds on the amount of memory and/or steps required by any algorithm that solves A. Establishing complexity bounds for a problem can seem extremely difficult if not impossible, but reducibility can nevertheless give us some insight with regards to the relative complexity of the problem.

Theorem 10.1. If $A \leq_{\mathrm{T}} B$ or $A \leq_m B$, then the following statements hold.

1. If B is solvable then so is A.
2. If A is unsolvable then so is B.

Proof. Since a mapping reducibility is a special case of Turing reducibility, we may assume $A \leq_{\mathrm{T}} B$.

For the first statement, suppose B is solvable via some algorithm $\mathcal{B}$. Let $\mathcal{R}$ denote the algorithm that Turing reduces A to B by making use of queries to a B-oracle. Note that $\mathcal{R}$ may not be a valid algorithm in the sense with which we are normally familiar. This is so because there may not be an algorithmic way to obtain the answers to the B-queries that appear in the computation. However, since B is solvable via $\mathcal{B}$, there *is* an algorithmic means for obtaining the answers, and we may replace each query of the form query(b), where b is an instance of B, with a call to algorithm $\mathcal{B}$ on input b. Therefore, $\mathcal{R}$ together with algorithm $\mathcal{B}$ that is used to answer B-queries may be combined to define an algorithm in the normal sense.

Finally, notice that the second statement is just the contrapositive of the first, an thus is also a true statement. □

Theorem 10.2. If $A \leq_m^p B$, then the following statements hold.

1. If B is solvable in $O(p(m))$ steps, for some polynomial p, where m is the size parameter for B, then A is solvable in $O(r(n))$ steps, for some polynomial $r(n)$, where n is the size parameter for A.
2. If A cannot be solved in $O(r(n))$ steps, for any polynomial $r(n)$, then B cannot be solved in $O(p(m))$ steps for any polynomial $p(m)$.

Proof. As with Theorem 10.1, the second statement is the contrapositive of the first, and so it suffices to prove the first. To this end, assume B is solvable in $O(p(m))$ steps via some algorithm $\mathcal{B}$. Since $A \leq_m^p B$, there is a map $f : A \to B$ that is computable in $O(q(n))$ steps for some polynomial $q(n)$, where n is the size parameter for A. Moreover, the solution to instance a of A equals the solution to instance $f(a)$ of B. Therefore, an algorithm for solving A first computes instance $f(a)$ via the algorithm that computes f, and then applies algorithm $\mathcal{B}$ to instance $f(a)$. Now, since $f(a)$ is computable in $O(q(n))$ steps, we may assume that the size of $f(a)$ is $m = O(q(n))$ bits, i.e. each algorithm step can construct at most $O(1)$ bits of the output. Thus, $\mathcal{B}$ runs on input $f(a)$ whose size is $m = O(q(n))$ bits which means the algorithm's running time is $O(p(q(n)))$. Thus, the total running time for solving instance a is

$$O(q(n)) + p(q(n))) = O(p(q(n))),$$

and so we may set $r(n) = q(p(n))$. □

Example 10.3. Suppose f map reduces A to B and is computable in $O(n^3)$ steps. Furthermore, suppose algorithm $\mathcal{B}$ solves an instance of B in $O(m^4)$ steps. Determine the running time of the following algorithm that solves A.

Algorithm for Solving A

Input: instance a of A.

Output: solution to a.

Compute $b = f(a)$ which is an instance of B.

Return $\mathcal{B}(b)$.

Exercises

1. Consider the following functions.

```
//Turing reduces A to B
Boolean solve_A_with_B(int n)
{
    //Solve instance n of A by making B-queries
    return query_B(n*n) || query_B(n+6);
}


//Turing reduces B to C
Boolean solve_B_with_C(int n)
{
    //Solve instance n of B by making C-queries
    return !query_C(n+8) && query_C(5*n);
}
```

   Implement a third function `solve_A_with_C` that is a witness to $A \leq_{\mathrm{T}} C$. Note: your function must take an instance n of A and return a Boolean decision that uses logic and is allowed to only make C-queries.

2. Use the previous exercise as inspiration for proving the general result that $\leq_{\mathrm{T}}$ reducibility relation is transitive. In other words, if $A \leq_{\mathrm{T}} B$ and $B \leq_{\mathrm{T}} C$, then $A \leq_{\mathrm{T}} C$.

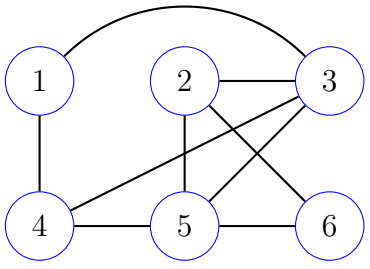
3. Recall the `Even` and `Odd` decision problems from Example 3.1. Which of the following functions provides a mapping reduction from `Even` to `Odd`?

   a. $f(n) = n^2$

   b. $f(n) = 2n + 5$

   c. $f(n) = 3n - 7$

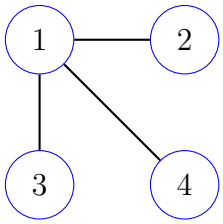
4. Draw the complement of the graph below.



5. For the mapping reduction f : `Max IS` $\rightarrow$ `Max Clique` in Example 3.2 defined by $f(G) = \overline{G}$, explain why f may also be used to reduce `Max Clique` to `Max IS`.

6. Given the instance $S = \{16, 21, 23, 25, 38, 47, 55, 73\}$ of `Set Partition`, provide the instance of `SS` to which it reduces via the mapping $f(S) = (S, t = M/2)$.

7. The graph G shown below represents an instance of the `Hamilton Path` decision problem. Compute $f(G)$, where f is the mapping reduction from `HP` to `LPath` described in Example 4.10. Verify that the mapping is correct in the sense that the decision for G is equal to the decision for $f(G)$. Explain.



8. Repeat Example 5.3 but with the following instances of `Subset Sum` to `Set Partition`.

   a. $(\{3, 7, 11, 29, 44, 53, 66, 81\}, t = 86)$

   b. $(\{3, 7, 11, 29, 44, 53, 66, 81\}, t = 147)$

   c. $(\{3, 7, 11, 29, 44, 53, 66, 81\}, t = 177)$

9. An instance of decision problem `Set Tri-Partition (STP)` is a set of nonnegative integers S, and the problem is to decide if there exist subsets $A_1, A_2, A_3 \subseteq S$ for which i) $A_1 \cup A_2 \cup A_3 = S$, ii) $A_i \cap A_j = \emptyset$, for $i \neq j$, and iii)

$$\sum_{a \in A_1} a = \sum_{a \in A_2} a = \sum_{a \in A_3} a.$$

   Show that the mapping $f : \texttt{STP} \to \texttt{SS}$ defined by $f(S) = (S, M/3)$ where

$$M = \sum_{s \in S} s,$$

   is *not* a valid mapping reduction. Hint: provide a negative instance of `STP` and show that f maps it to a positive instance.

10. Recall the contraction mapping f from `VC` to `HVC` provided in Example 5.5. Suppose $G = (V, E)$ is a simple graph with $|V| = 135$. Then if $f(G, k = 39) = G'$, then describe the relationship between G' and G.

11. Repeat the previous exercise, but now assume $|V| = 152$ and $k = 100$.

12. The `Half Clique` decision problem is the problem of deciding if a simple graph $G = (V, E)$ has a Clique of size $|V|/2$. Provide an embedding reduction f from `Half Clique` to `Clique`.

13. Provide a contraction reduction f from `Clique` to `Half Clique`. Defend your answer, meaning prove that (G, k) is a positive instance of `Clique` iff $f(G, k)$ is a positive instance of `Half Clique`.

14. Consider `3SAT` instance

$$\mathcal{C} = \{c_1 = (x_1, \overline{x}_2, \overline{x}_3), c_2 = (\overline{x}_1, x_2, x_3), c_3 = (x_1, x_2, x_3)), c_4 = (\overline{x}_1, \overline{x}_2, x_3)\},$$

   consider the polynomial-time mapping reduction $f(\mathcal{C}) = (G, k)$ from `3SAT` to `Clique` described in Theorem 7.1.

a. How many edges does G have?

b. What is the value of k? Does G have a k-clique? Explain and provide one if your answer is "yes".

15. For the polynomial-time reduction f from `3SAT` to `Clique` described in Theorem 7.1, if an instance $\mathcal{C}$ of `3SAT` has 536 clauses and 243 variables, then, given $(G, k) = f(\mathcal{C})$, how many vertices does G have? Provide a good upper bound on G's size (i.e. number of edges). What is the value of k?

16. For the polynomial-time reduction f from `3SAT` to `Clique` described in Theorem 7.1, how does the reduction change if we reduce from `4SAT` instead of `3SAT`. Repeat the previous problem but with the reduction coming from an instance of `4SAT`.

17. Consider `3SAT` instance

$$\mathcal{C} = \{c_1 = (\overline{x}_1, \overline{x}_2, \overline{x}_3), c_2 = (\overline{x}_1, x_2, \overline{x}_3), c_3 = (x_1, \overline{x}_2, x_3)), c_4 = (x_1, \overline{x}_2, x_3)\},$$

consider the polynomial-time mapping reduction $f(\mathcal{C}) = (S, t)$ from `3SAT` to `Subset Sum` described in Theorem 7.3.

a. Draw S and t as a table of values.

b. Does S have a subset that sums to t? Explain and provide one if your answer is "yes".

18. For the reduction f : `3SAT` $\rightarrow$ `SS` from `3SAT` to `Subset Sum` provided in Theorem 7.3, if an instance $\mathcal{C}$ of `3SAT` has 275 clauses and 57 variables, then how many integers does the set S have, where $f(\mathcal{C}) = (S, t)$? What is the value of the target integer t?

19. For the reduction f : `3SAT` $\rightarrow$ `SS` from `3SAT` to `Subset Sum` provided in Theorem 7.3, suppose an instance $\mathcal{C}$ of `3SAT` has 57 clauses and 10 variables. Assuming $f(\mathcal{C}) = (S, t)$, what is the size of the smallest subset of S that could possibly sum to the target value t. Explain. What is the greatest size? Explain.

20. For bipartite graph $G = (U, V, E)$ we have $U = \{u_1, u_2, u_3, u_4\}$, $V = \{v_1, v_2, v_3, v_4\}$, and

$$E = \{(u_1, v_1), (u_1, v_2), (u_1, v_4), (u_2, v_1), (u_2, v_3),$$

$$(u_2, v_4), (u_3, v_1), (u_3, v_3), (u_4, v_1), (u_4, v_3)\}.$$

a. Draw G.

b. Does G have a (non-maximum) maximal matching M of size 1? size 2? size 3?

c. For each `yes` answer to the previous part, draw the residual network G'_{f_M} associated with the matching, and provide an alternating path P in G'_{f_M}. Use the alternating path to find an increment of M.

21. At a school ice-cream party there are five dixie cups of ice cream that remain to be served. Each cup has a different flavor: vanilla, chocolate, cherry, rocky road, and mint and chip. There are five children who have yet to be served: Abe, Ben, Cris, Dan, and Eva. The ice-cream preferences of these children are shown below.

Child	Vanilla	Chocolate	Cherry	Rocky Road	Mint & Chip
Abe	X		X		X
Ben		X			
Cris		X		X	
Dan		X			X
Eva			X	X	

In a rush to get their ice cream, Abe grabbed the cherry, Cris the chocolate, Dan the mint and chip, and Eva the rocky road. This left Ben with a (vanilla) flavor that he does not like, and which he refused to eat. Show how the maximum-matching algorithm can be used to increase the current matching (of four children to four ice creams that they prefer) to a matching of size five, in which each child will be assigned an ice cream that he or she prefers.

22. Recall the mapping reduction from `Max IS` to `Max Clique` described in Example 3.2. Show that this reduction can be computed in polynomial-time by providing a big-O expression that gives the running time for computing the reduction. Defend your answer.

23. Recall the mapping reduction from `Subset Sum (SS)` to `Set Partition (SP)` described in Section 5. Show that this reduction can be computed in polynomial-time by providing a big-O expression that gives the running time for computing the reduction. Defend your answer.

Exercise Solutions

1. We have the following function that proves $A \leq_{\mathrm{T}} C$.

```
//Turing reduces A to C
Boolean solve_A_with_C(int n)
{
    //Solve instance n of A by making C-queries
    return (!query_C((n*n)+8) && query_C(5*(n*n))) ||
         (!query_C((n+6)+8) && query_C(5*(n+6)));
}
```

2. Suppose $A \leq_{\mathrm{T}} B$. Then there is an algorithm $\mathcal{A}_{AB}$ that solves an instance of A by making queries to a B-oracel. Moreover, since $B \leq_{\mathrm{T}} C$, there is also an algorithm $\mathcal{A}_{BC}$ that solves an instance of B by making queries to a C-oracle.

   We now describe an algorithm $\mathcal{A}_{AC}$ that solves instances of A by making queries to a C-oracle. This algorithm is obtained by modifying $\mathcal{A}_{AB}$ as follows. For each B-query step query(y), where y is an instance of B, we replace this B-query step with a function call to $\mathcal{A}_{BC}(y)$, which is the answer returned by $\mathcal{A}_{BC}$ on input y. We may think of $\mathcal{A}_{BC}(y)$ as a function call that is being made within the body of $\mathcal{A}_{AB}$. Of course, the $\mathcal{A}_{BC}$ function has its own body of source code, which is now part of the $\mathcal{A}_{AC}$ code base. After modifying $\mathcal{A}_{AB}$ in this manner, notice that the only query steps in $\mathcal{A}_{AC}$ are found in the inserted $\mathcal{A}_{BC}$ code, and are of the form query(z), where z is a problem instance of C. In other words, all queries are to a C-oracle. Hence, $\mathcal{A}_{AC}$ is an algorithm that Turing reduces A to C.

3. f : `Even` $\rightarrow$ `Odd` map reduces `Even` to `Odd` iff it maps evens to odds and odds to evens (why?).

   a. $f(n) = n^2$. No, since f maps evens to evens and odds to odds. For example $f(2) = 4$.

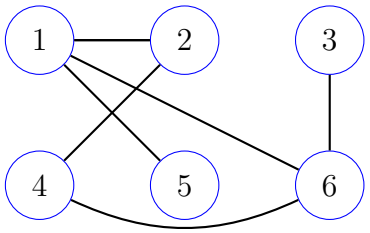
   b. $f(n) = 2n + 5$. No, since f maps evens to odds, but maps odds to odds. For example, $f(3) = 11$.

   c. $f(n) = 3n - 7$. Yes. If n is even, then $n = 2k$ for some integer k. Thus

   $$3n - 7 = 3(2k) - 8 + 1 = 6k - 8 + 1 = 2(3k - 4) + 1,$$

   which is an odd number. Similarly, if n is odd, $3n - 7$ is even.

4. The complement graph is shown below.



5. First, notice that `Max Clique` and `Max IS` both have the exact same problem instances, namely the set of all simple graphs. Thus, the domain and codomain of f are identical. Moreover, since $f(G) = \overline{G}$, G will have a clique of size k iff $f(G)$ has an independent set of size k. Therefore, f may also be used to reduce `Max Clique` to `Max IS`.

6. $f(S) = (S, t = 149)$

7. $f(G) = (G, k = n - 1) = (G, k = 3)$. For this instance, the mapping is correct since G is a negative instance of `HP` which is equivalent to saying that it does not have a simple path of length 3. Furthermore G is negative for `HP` because three of G's vertices have degree 1, but a simple path of length three requires at least two vertices that have degree at least 2.

8. Repeat Example 5.3 but with the following instances of `Subset Sum` to `Set Partition`.

   a. $S = \{3, 7, 11, 29, 44, 53, 66, 81, 122\}$

   b. $S = \{3, 7, 11, 29, 44, 53, 66, 81\}$

   c. $S = \{3, 7, 11, 29, 44, 53, 60, 66, 81\}$

9. Consider the `STP` instance $S = \{6, 12\}$. Then $f(S) = (S, k = 18/3 = 6)$ is a positive instance of `Subset Sum` via $A = \{6\}$, but S is a *negative* instance of `STP` (why?).

10. Since $k = 39 < 135/2 = 67.5$, we have $f(G, k = 39) = G'$, where G' is the graph G with the addition of J triangles, where J satisifes

$$\frac{39 + 2J}{135 + 3J} = \frac{1}{2}$$

    which implies $J = 57$. Thefore, G' is the same graph G, but with the addition of 57 distinct triangles. The addition of these triangles will make it so that G' has a half vertex cover iff G has a vertex cover of size 39.

11. Since $k = 100 > 152/2 = 76$, $f(G, k = 100) = G'$, where G' is the graph G with the addition of $2(100) - 152 = 48$ new isolated vertices.

12. Let $G = (V, E)$ be a simple graph. Then $f(G) = (G, k = |V|/2)$, i.e. G is a positive instance of `Half Clique` iff it has a clique of size $|V|/2$ iff $(G, k = |V|/2)$ is a positive instance of `Clique`.

13. The contraction reduction from `Clique` to `Half Clique` is similar to the one given in Example 5.5. Let $G = (V, E)$ be a simple graph and $k \geq 0$ be a nonnegative integer between 1 and $n = |V|$. If $k = n/2$, then $f(G, k) = G$ since (G, k) would then be an actual `Half Clique` problem instance. Now suppose $k < n/2$. Then $f(G, k) = G'$ where G' is formed by adding J additional vertices to G and placing edges between them so that they form a J-clique C_J. Futhermore, we also add an edge between each vertex in C_J and each vertex in V. Therefore, G will have a k clique iff G' has a $k + J$ clique. Moreover, this $k + J$-clique will be a half clique for G' iff

$$k + J = \frac{1}{2}(n + J) \Leftrightarrow J = n - 2k.$$

    Therefore, we require that $J = n - 2k$. Finally, if $k > n/2$, then $f(G, k) = G'$ where G' is formed by adding J additional *isolated* vertices to G Therefore, G will have a k clique iff G' has a k clique. Moreover, this k-clique will be a half clique for G' iff

$$k = \frac{1}{2}(n + J) \Leftrightarrow J = 2k - n.$$

14. We must count the pairs of consistent vertices between vertex groups $c_1 - c_2$, $c_1 - c_3$, $c_1 - c_4$, $c_2 - c_3$, $c_2 - c_4$, $c_3 - c_4$. The number of pairs of consistent vertices sums to

$$6 + 7 + 7 + 8 + 8 + 7 = 43$$

edges total. Also, $k = |[\mathcal{C}| = 4$ and G does have a 4-clique since $\alpha = (x_1 = 1, x_2 = 1, x_3 = 1)$ satisfies $\mathcal{C}$ (a positive instance of 3SAT must map to a positive instance of `Clique`). One such clique is $C = \{x_1, x_2, x_1, x_3\}$, where the i th literal listed in the set comes from clause c_i, $i = 1, 2, 3, 4$.

15. We have $f(\mathcal{C}) = (G = (V, E), k)$ where $k = 536$, $|V| = 536 \times 3 = 1608$ vertices, and at most

$$\frac{9(536)(535)}{2} = 1,290,420$$

edges (yikes!).

16. If `4SAT` were used instead of `3SAT`, then every vertex group would have 4 (instead of 3) vertices. Then have $f(\mathcal{C}) = (G = (V, E), k)$ where $k = 536$, $|V| = 536 \times 4 = 2144$ vertices, and at most

$$\frac{16(536)(535)}{2} = 2,294,080$$

edges (yikes!).

17. We have the following table that describes the members of S and t.

	1	2	3	c_1	c_2	c_3	c_4
y_1	1	0	0	0	0	1	1
z_1	1	0	0	1	1	0	0
y_2		1	0	0	1	0	0
z_2		1	0	1	0	1	1
y_3			1	0	0	1	1
z_3			1	1	1	0	0
g_1				1	0	0	0
h_1				1	0	0	0
g_2					1	0	0
h_2					1	0	0
g_3						1	0
h_3						1	0
g_4						0	1
h_4						0	1
t	1	1	1	3	3	3	3

Also, since $\alpha = (x_1 = 0, x_2 = 1, x_3 = 1)$ satisfies $\mathcal{C}$ and a positive instance of 3SAT must map to a positive instance of `Subset Sum`, (S, t) is a positive instance and

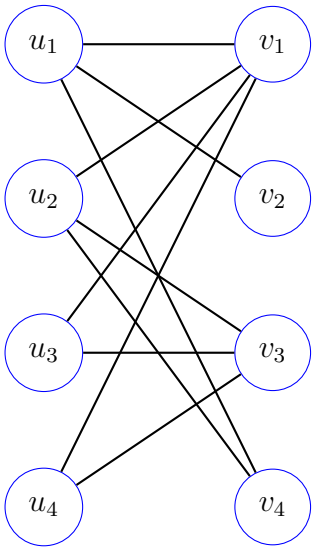
$$A = \{z_1, y_2, y_3, g_1, h_1, g_2, g_3, h_3, g_4, h_4\}$$

sums to $t = 1,113,333$ (verify!).

18. $|S| = 2(57) + 2(275) = 664$, and $t = 1 \cdots 13 \cdots 3$ has 57 1's and 275 3's.

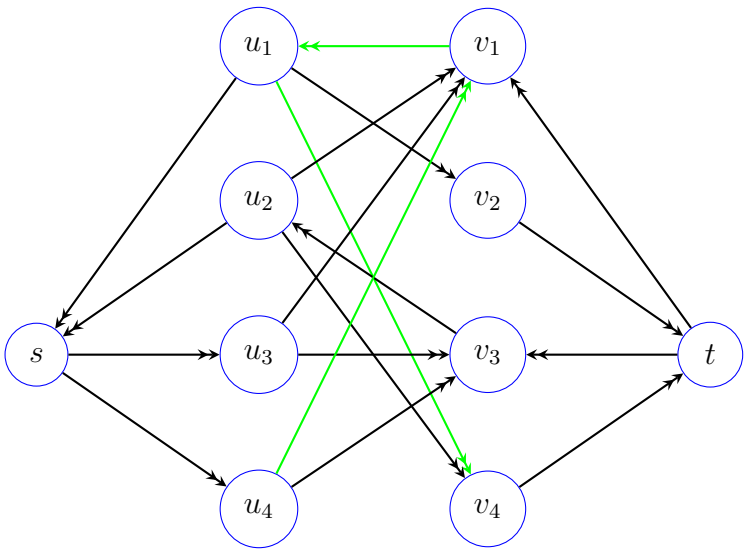
19. The smallest subset of S that could possibly sum to t has a size equal to 10, since either y_i or z_i must be selected (but not both) for $i = 1, \ldots, 10$. This corresponds with every literal of every clause being satisfied by some assignment. On the other hand, the largest subset could have size equal to $2(57) + 10 = 124$. This would be the case where, in addition to selecting a, y_i or z_i, both filler numbers would also be selected for every clause. This corresponds with exactly one literal of every clause being satisfied by some assignment.

20. a. Below is a graph of $G = (U, V, E)$

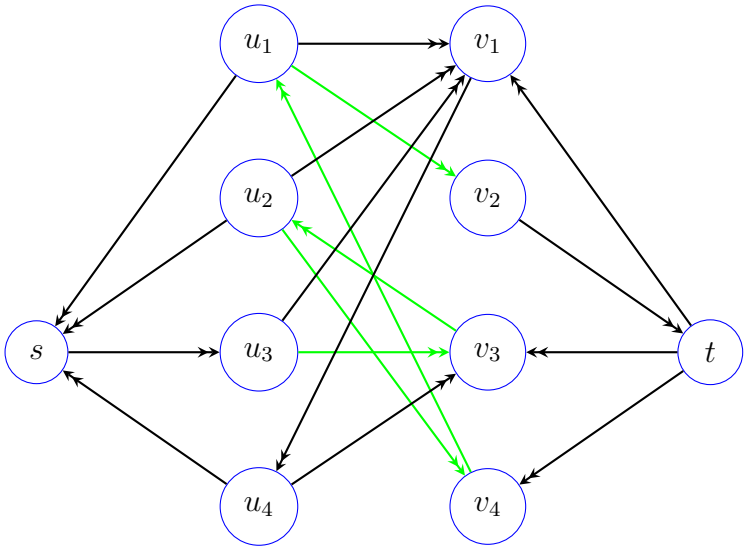


b. G does not have a size-1 maximal matching since, for any edge e, there is an edge e_2 that does not share any vertices with e. However, $M_2 = \{(u_1, v_1), (u_2, v_3)\}$ is a maximal matching of size 2, since u_1 and u_2 are the only vertices incident with v_2 and v_4. Then $P = u_4, v_1, u_1, v_4$ is an alternating path in $G'_{F_{M_2}}$, and $M_3 = P \oplus M_2 = \{(u_1, v_4), (u_4, v_1), (u_2, v_3)\}$ is a maximal matching of size 3.

c. $G'_{F_{M_2}}$ is shown below with an alternating path P drawn in green. This yields increment $M_3 = P \oplus M_2 = \{(u_1, v_4), (u_4, v_1), (u_2, v_3)\}$.



$G'_{F_{M_3}}$ is shown below with an alternating path P drawn in green. This yields increment $M_4 = P \oplus M_3 = \{(u_1, v_2), (u_2, v_4), (u_3, v_3), (u_4, v_1)\}$.

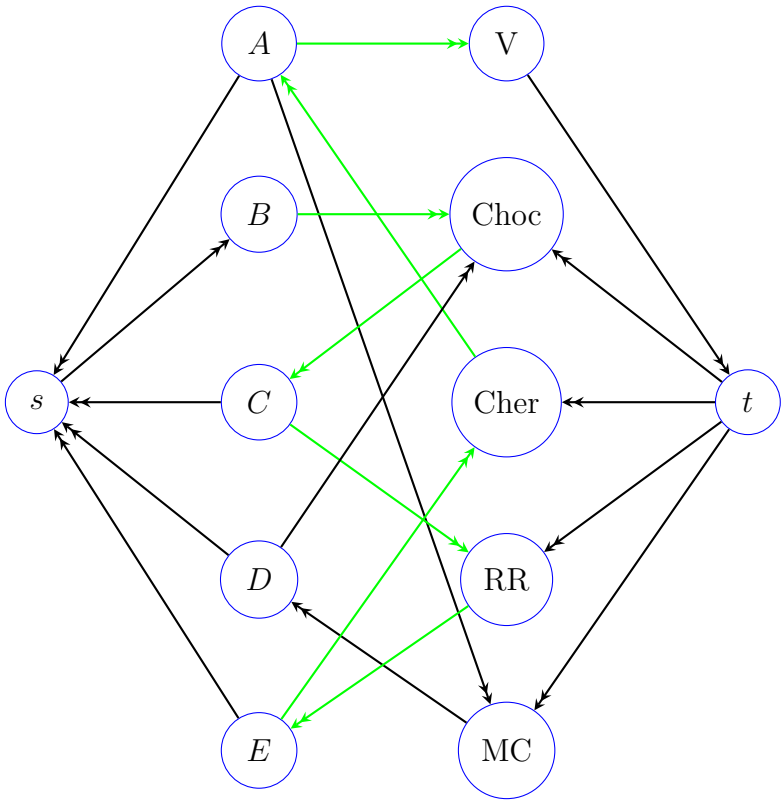


21. Let $G = (U, V, E)$ be the bipartite graph whose U set represents the set of children, and V set represents the set of ice-cream flavors. The $(u, v) \in E$ iff child u likes ice cream v. The children rushing for their ice cream resulted in the matching

$$M = \{(A, \text{Cher}), (C, \text{Choc}), (D, \text{MC}), (E, \text{RR})\}.$$

The residual network G'_{F_M} below shows an alternating path P (in green) for which

$$M^* = P \oplus M = \{(A, \text{V}), (B, \text{Choc}), (C, \text{RR}), (D, \text{MC}), (E, \text{Cher})\}.$$



22. Recall the mapping reduction from `Max IS` to `Max Clique` described in Example 3.2. Use high-level pseudocode (similar to the style used throughout this chapter) to describe an algorithm that computes the reducing function $f(G) = \overline{G}$ and requires a polynomial number of steps in the size parameters for `Max IS`. Provide a good big-O bound on the running time. Conclude that `Max IS` $\leq_m^p$ `Max Clique`.

23. Recall the mapping reduction from `Subset Sum (SS)` to `Set Partition (SP)` described in Section 5. Use high-level pseudocode (similar to the style used throughout this chapter) to describe an algorithm that computes the reducing function f and requires a polynomial number of steps in the size parameters for `SS`. Provide a good big-O bound on the running time. Conclude that `SS` $\leq_m^p$ `SP`.