**Probabilistic Analysis of Algorithms**

Alan M. Frieze* Bruce Reed†

# 1 Introduction

Rather than analyzing the worst case performance of algorithms, one can investigate their performance on typical instances of a given size. This is the approach we investigate in this paper. Of course, the first question we must answer is: what do we mean by a typical instance of a given size?

Sometimes, there is a natural answer to this question. For example, in developing an algorithm which is typically efficent for an NP-complete optimization problems on graphs, we might assume that an $n$ vertex input is equally likely to be any of the $2^{\binom{n}{2}}$ labelled graphs with $n$ vertices. This allows us to exploit any property which holds on almost all such graphs when developing the algorithm.

There is no such obvious choice of a typical input to an algorithm which sorts $n$ numbers $x_1, ..., x_n$ for, e.g., it is not clear how big we want to permit the $x_i$ to become. One of many possible approaches is to impose the condition that each number is drawn uniformly from $[0, 1]$. Another is to note that in analyzing our algorithm, we may not need to know the values of the variables but simply their relative sizes. We can then perform our analysis assuming that the $x_i$ are a random permutation of $y_1 < y_2 < ... < y_n$ with each permutation equally likely.

More generally, we will choose some probability distribution on the inputs of a given size and analyze the performance of our algorithm when applied to a random input drawn from this distribution. Now, in general, probability distributions are complicated objects which must be formally described and analyzed using much messy measure theory. Fortunately, we will be concerned only with relatively simple distributions which will be much easier to deal with.

We often consider *finite distributions* in which our probability space is a finite set $S$, and for each $x \in S$ there is a $p_x$ such that the $\Sigma_{x \in S} p_x = 1$ and the probability that the outcome is $x$ is $p_x$. If

*Department of Mathematical Sciences, Carnegie-Mellon University, Pittsburgh, PA 15213. Supported in part by NSF grant CCR9530974. E-mail: af1p@andrew.cmu.edu.

†Equipe Combinatoire, CNRS, Univ. de Paris VI, 4 Place Jussieu, Paris 75005, France. E-mail: reed@ecp6.jussieu,fr

all the $p_x$ are the same then we are choosing a *uniform* member of $S$. For example, we discussed above choosing uniformily a random labelled graph on $n$ vertices.

We may also consider choosing reals uniformly in $[a, b]$. Thus the probability our random real is between $c$ and $d$ for $a \leq c < d \leq b$ is $\frac{d-c}{b-a}$.

Alternatively, we may consider analyzing probability distributions by imposing conditions on the random objects chosen without specifying any further the underlying distribution. One example of such a *distribution independent* analysis was mentioned earlier when we suggested studying sorting under the assumption that all $n!$ permutations of $n$ numbers are equally likely to be the input.

Finally, we may consider combining the above three possibilities. For example, we may consider a uniformly chosen graph on $n$ vertices whose edges have been assigned uniform random weights from $[0, 1]$, or a set $S$ of random vectors in $R^m$ where each vector consists of $m$ independent uniform elements of $[0, 1]$.

Focusing on these simple distributions allows us to dispense with the development of a rigorous measure theoretical foundation of probability theory. It is also quite natural.

One of our goals in this paper is to develop exact algorithms which work efficiently on the overwhelming majority of random inputs. A related goal is to try and find algorithms whose expected running time is small. We examine these approaches in Sections 2 and 3. A different technique is to consider algorithms which are guaranteed to run quickly but do not necessarily find the optimal solution, and show they are typically optimal, very close to optimal, or at least reasonably close to optimal. This is the approach taken in Sections 4 and 5.

Alternatively, we can show that an algorithm almost always behaves poorly on random instances. For example, we might prove that an algorithm almost always takes exponential time. This is a much more damning condemnation of its performance than the pathological examples constructed to provide lower bounds on worst-case complexity. We discuss this approach in Section 6. Finally, we note that how an algorithm performs on a random input depends heavily on the probability distribution we are using. In Section 7, we compare the analysis of various probability distributions for some specific problems.

We stress that we are interested in providing the reader with a gentle introduction to some of the most important topics in this area. Our survey is neither comprehensive nor up to date. Readers may turn to the survey articles [45], and the books [28], [80], [85] for more in-depth discussions of this area.

## 1.1 Some Basic Notions

We begin with two simple but powerful probabilistic tools.

**The Chernoff/Hoeffding Bounds** Suppose $X$ is the sum of $n$ independent random variables $X_1, X_2, \ldots, X_n$ where $0 \leq X_j \leq 1$ and $\mathbf{E}(X_j) = \mu_j$ for $1 \leq j \leq n$. Let $\mu = (\mu_1 + \mu_2 + \cdots + \mu_n)/n$ so that $\mathbf{E}(X) = n\mu$. Then for $0 \leq a \leq 1$:

$$\mathbf{Pr}(|X - \mathbf{E}(X)| > an\mu) \leq 2e^{-a^2 n\mu/3}.$$

This is one of many inequalities which bound the extent to which a variable is concentrated around its expected value. Chapter 7 of this volume is dedicated to the study of such inequalities.

**The First Moment Method/Markov Inequality** If $X$ is a random non-negative integer valued variable then

$$\mathbf{Pr}(X > 0) \leq \mathbf{E}(X)$$

(Proof: $\mathbf{Pr}(X > 0) = \Sigma_{i=1}^{\infty} \mathbf{Pr}(X = i) \leq \Sigma_{i=1}^{\infty} i\mathbf{Pr}(X = i) = \mathbf{E}(X)$).
Moreover, $\mathbf{E}(X)$ is often easier to compute than $\mathbf{Pr}(X > 0)$. If this is the case, then we may compute $\mathbf{E}(X)$ and use it as a bound on $\mathbf{Pr}(X = 0)$. This technique is known as the First moment method.

We say that a property defined in terms of $n$ holds **whp** if it holds with probability $1 - o(1)$ as $n \to \infty$.

By $G_{n,p}$ we mean a random graph with vertex set $V_n = \{1, ..., n\}$ where each edge is present with probability $p$ independently of the presence of the other edges. Thus, for each graph $H$ with vertex set $V_n$ and $m$ edges the probability that $G_{n,p} = H$ is $p^m(1-p)^{\binom{n}{2}-m}$. In particular, $G_{n,\frac{1}{2}}$ is a uniformly chosen random graph with vertex set $V_n$.

We note that the expected number of edges in $G_{n,p}$ is $p\binom{n}{2}$. Further, the Chernoff bounds can be used to show that unless $p = O(1/n^2)$, $|E(G_{n,p}|$ is **whp** $(1 - o(1))p\binom{n}{2}$. Thus, if we analyze $G_{n,p}$, then typical graphs have about $p\binom{n}{2}$ edges. $G_{n,m}$ is the random graph on $n$ vertices whose edge set $E_{n,m}$ is a uniformly chosen random set of $m$ of the $\binom{n}{2}$ unordered pairs contained within $\{1, .., n\}$.

Finally, we note that if we have an algorithm $A$ for an optimization problem and we run it on a random instance $I$ of size $n$ drawn from some probability distribution, then the running time of this algorithm on this instance, $R_{A,n}(I)$, is a random variable which depends on $I$. We let its expected value be $r_{A,n}$. The expected running time of algorithm $A$ with respect to the specified distribution is a function $ER_A$ such that $ER_A(n) = r_{A,n}$.

# 2 Exact Algorithms for Hard Problems

NP-complete problems are natural candidates for probabilistic analysis, as the traditional worst-case approach has failed to provide efficent algorithms for such problems. In this section, we focus on two such problems, Edge Colouring, and Hamilton Cycle. We shall also discuss Graph Isomorphism, another problem which although not known to be NP-complete, also is not known

to be solvable in polynomial time. As we shall see, it makes little sense to speak of approximation algorithms for any of these problems. Thus, the failure to find efficient algorithms to solve them means that from a traditional viewpoint we are completely at sea. Our first step is to find efficient algorithms which solve these problems **whp** on uniform random instances, we then present algorithms which have polynomial expected running time.

Some may criticise as unrealistic the assumption that a typical input is a uniformly chosen graph. However, this is no more unrealistic then the belief that studying the pathological examples constructed in NP-completeness proofs yields information about typical instances. Furthermore, a standard paradigm for constructing algorithms which run in polynomial time **whp** (though by no means the only one), is to provide an algorithm which works provided that the input graph has a certain structure and then prove that $G_{n,\frac{1}{2}}$ has the required structure **whp**. Such proofs are valuable because they add to our understanding of what it is that makes the problem difficult.

## 2.1 Algorithms which almost always succeed

### 2.1.1 Hamilton Cycles

A Hamilton cycle in a graph $G$ is one passing through all its vertices. Determining if a graph has a Hamilton cycle was one of the first six NP-complete problems reduced to SAT by Karp in his seminal paper [**?**]. In this section we show that $G_{n,\frac{1}{2}}$ has a Hamilton cycle **whp** and present a polynomial-time algorithm which **whp** constructs such a cycle. This is not difficult as there are a large number of random edges.

**Definition:** We call a graph, *tractable*, if the following conditions hold:

(i) every vertex has between $\frac{n}{2} - \frac{n}{50}$ and $\frac{n}{2} + \frac{n}{50}$ neighbours.

(ii) for every pair $\{u, v\}$ of vertices, we have: $\frac{3n}{4} - \frac{n}{50} \leq |N(u) \cup N(v)| \leq \frac{3n}{4} + \frac{n}{50}$

(iii) for every triple $\{u, v, w\}$ of vertices, we have:

$$\frac{7n}{8} - \frac{n}{50} \leq |N(u) \cup N(v) \cup N(w)| \leq \frac{7n}{8} + \frac{n}{50}$$

We need:

**Lemma 1** *$G_{n,\frac{1}{2}}$ is tractable* **whp**.

**Proof** For each pair of vertices $\{u, v\}$ of $G_{n,\frac{1}{2}}$, $|N(v) \cup N(u) - u - v|$ is the sum of $n-2$ independent random variables each of which is 1 with probability $\frac{3}{4}$ and 0 with probability $\frac{1}{4}$.

Thus, we can show (ii) holds **whp** using the Chernoff bounds. Similar techniques apply for (i) and (iii), we leave the details to the reader. □

We now present a polynomial-time algorithm for constructing a Hamilton cycle in a tractable graph, which by the above lemma works **whp** on $G_{n,\frac{1}{2}}$. The algorithm has three phases.

**Phase 1:** Path Construction

Construct a path $P$ by iteratively applying the following two rules, until this is no longer possible.

(i) If some vertex $x$ not on $P$ sees an endpoint $v$ of $P$, add the edge $xv$ to $P$,

(ii) if there are vertices $x \notin P$,$y, z \in P$ such that $P = vP'yzP''$ and $xy, vz \in E(G)$ then replace $P$ by the path $xyP'vzP''$

We leave it as an exercise for the reader to show that in a tractable graph with $n \geq 20$ vertices, the final path has at least $\frac{7n}{8} - \frac{n}{50}$ vertices.

**Phase 2:** Cycle Construction

Construct a path $C$ by applying one of the following two rules.

(i) if there are vertices $x, y \in P$, such that $P = vP'xyP''w$ and $vy, wx \in E(G)$ then let $C$ be the cycle $wxP'vyP''w$

(ii) if there are vertices $x, y \in P$,such that $P = vP'xyP''w'w$ and $vy, w'x \in E(G)$ then let $C$ be the cycle $w'xP'vyP''w'$

We leave it as an exercise for the reader to show that in a tractable graph with $n \geq 20$ vertices, this phase is always possible. We note that $|C| \geq \frac{7n}{8} - \frac{n}{50} - 1$.

**Phase 3:** Cycle Extension

We add the vertices of $V - C$ to $C$, one or two at a time, until $V(C) = V$, according to the following three rules:

(i) If some vertex $x$ not on $P$ sees two consecutive vertices $y$ and $z$ of $C$ then replace $C$ by $C - yz + yx + xz$,

(ii) if there are adjacent vertices $x, y \notin P$, and consecutive vertices $u, v$ of $C$ such that $ux, yv \in E(G)$ then replace $C$ by the $C - uv + ux + xy + yv$,

(iii) if there are vertices $x \notin C$ and vertices $y, z, a, b \in C$ such that $C = abP'yzP''a$ and $xa, xy, bz \in E(G)$ then replace $C$ by the cycle $axyP'bzP''a$.

We leave it as an exercise for the reader to show that in a tractable graph with $n \geq 50$ vertices, this step is always possible (Hint: If $V - C$ is not a stable set then we can apply (i) or (ii)).

It is easy to see that each phase of the algorithm can be implemented in $O(n^3)$ time, so it is indeed a polynomial-time algorithm as claimed.

**Exercise:** Show that the above algorithm can actually be implemented in $O(n^2)$ time on tractable graphs (which is linear in the number of edges).

### 2.1.2 Edge Colouring

An edge colouring of a graph $G$ is an assignment of colours to its edges so that no two edges which share an endpoint receive the same colour. I.e., each colour class is a *matching*, that is a graph all of whose vertices have degree one. Clearly, if a graph has maximum degree $\Delta$ then it has no edge colouring using fewer than $\Delta$ colours. Vizing proved that every such graph has a $\Delta + 1$ colouring. So determining the chromatic index of a graph boils down to determining if $G$ has a $\Delta$-colouring. Vizing [90] also proved that if the maximum degree vertices of $G$ form a *stable set*, i.e. no edge of $G$ links two vertices of maximum degree, then $G$ has a $\Delta$ colouring. Berge and Fournier [9] developed a polynomial time algorithm for constructing a $\Delta+1$ colouring of $G$. The algorithm provides a $\Delta$ colouring provided the vertices of maximum degree in $G$ form a stable set. In contrast Holyer[**?**] has shown that determining the chromatic number of a graph is NP-complete.

In this section, we present the following result due to Erdos and Wilson [38].

**Theorem 1** *$G_{n,\frac{1}{2}}$ has a unique vertex of maximum degree* **whp**.

Thus, we obtain:

**Corollary 1** *Berge and Fournier's algorithm is a polynomial-time algorithm which edge colours $G_{n,\frac{1}{2}}$* **whp**.

**Proof of Theorem 1** To prove the theorem, we need to analyze the probability distribution on the degrees of the vertices in $G_{n,\frac{1}{2}}$. Now, the degree of a vertex in $G_{n,\frac{1}{2}}$ is the sum of $n-1$ variables each of which is 0 with probability $\frac{1}{2}$ and 1 with probability $\frac{1}{2}$. Thus, the expected degree of a vertex of $G_{n,\frac{1}{2}}$ is $\frac{n-1}{2}$ and

**1** *The probability that $v$ has degree $i$ is $\frac{\binom{n-1}{i}}{2^n}$.*

It follows easily (e.g. from the Chernoff bounds) that if we let $t = t(n)$ be the smallest integer such that $\mathbf{Pr}(d(v) > t) < \frac{1}{n^{\frac{4}{5}}}$ then provided $n$ is large enough, $\frac{n}{2} \leq t \leq \frac{n}{2} + \sqrt{n \log n}$, so using (1) we obtain:

**2** $\mathbf{Pr}(d(v) > t) \geq \frac{1}{2}\mathbf{Pr}(d(v) > t-1) > \frac{1}{2n^{\frac{4}{5}}}$.

Thus, we expect at least $\frac{n^{\frac{1}{5}}}{2}$ vertices of $G_{n,\frac{1}{2}}$ to have degree greater than $t$. So, the following result, which we prove in the next section is not surprising.

**3 Whp** *there is a vertex of* $G_{n,\frac{1}{2}}$ *whose degree exceeds* $t$.

Now, a simple but tedious First Moment calculation, using (1) allows us to show:

**4 Whp** *there is no* $i > t$ *such that two vertices of* $G_{n,\frac{1}{2}}$ *have degree* $i$.

Combining (3) with (4) yields the theorem, it remains only to prove (4).

To do so, we note that, by (1), for $i$ between $t$ and $t_0 = t + \frac{\sqrt{n}}{8\log n}$, we have:

$$\frac{\mathbf{Pr}(d(v)=i)}{\mathbf{Pr}(d(v)=t)} = \frac{t!(n-1-t)!}{i!(n-1-i)!} = 1 + o(1).$$

Thus,

$$\mathbf{Pr}(d(v) > t) > \sum_{i=t+1}^{t_0} \mathbf{Pr}(d(v)=i) > \frac{\sqrt{n}}{16\log(n)}\mathbf{Pr}(d(v)=t)$$

So, we obtain that $\mathbf{Pr}(d(v)=t) = O(n^{-1.3}\log n)$.

We can now bound the expected number of pairs of vertices $N$ in both of which have the same degree $i$ which exceeds $t$. Let $\hat{d}(v) = d_{G_{n,\frac{1}{2}}-u}(v)$ for $v \in V_n$. Then

$$\begin{aligned}\mathbf{Pr}(d(u)=d(v)=i) &\le \mathbf{Pr}(\hat{d}(v)\in\{i-1,i\})\mathbf{Pr}(\hat{d}(u)\in\{i-1,i\})\\ &= \mathbf{Pr}(\hat{d}(u)\in\{i-1,i\})^2 \le \mathbf{Pr}(\hat{d}(u)\in\{i-1,i,i+1\})^2\\ &\le 9\mathbf{Pr}(\hat{d}(u)=i-1)^2\end{aligned}$$

Hence,

$$\mathbf{E}(N) \le 9\binom{n}{2}\sum_{i=t}^{n-1}(\mathbf{Pr}(d(v)=i))^2$$

$$\le 9\binom{n}{2}\sum_{i=t}^{t+\lfloor 3\sqrt{n\log n}\rfloor -1}(\mathbf{Pr}(d(v)=i))^2 + 9\binom{n}{2}\sum_{i=t+\lfloor 3\sqrt{n\log n}\rfloor}^{n-1}(\mathbf{Pr}(d(v)=i))^2$$

$$\le 30\binom{n}{2}\sqrt{n\log n}(\mathbf{Pr}(d(v)=t))^2 + 9\binom{n}{2}\sum_{i=t+\lfloor 3\sqrt{n\log n}\rfloor}^{n-1}(\mathbf{Pr}(d(v)>i))^2$$

Applying, the Chernoff bound and our bound on the probability that $d(v)=t$, we obtain

$$\mathbf{E}(N) = O(n^{-.6}(\log n)^2) + O(n^{-4}) = o(1).$$

Thus, the probability that for some $i \geq t$ there are two vertices of degree $i$ is also $o(1)$, i.e. (4) holds. □

A similar but messier First Moment computation yields the following result which we state without proof as we need it later:

**5** *For $j < \sqrt{n}$, the probability that there are $j$ disjoint pairs of vertices $\{x_1, y_1\}, ..., \{x_j, y_j\}$ such that for some $d_j > t$, $d_j = d(x_j) \leq d(y_j) \leq d_j + 4$ is $O(n^{\frac{-j}{20}})$*

As we discuss in Section 2.3, Frieze, Jackson, McDiarmid and Reed [44] showed that the probability that $G_{n,\frac{1}{2}}$ does not have a $\Delta$-edge colouring is between $(n^{-c_1 n})$ and $(n^{-c_2 n})$ for some two positive constants $c_1$ and $c_2$ ( and $n \geq 3$).

### 2.1.3 Graph Isomorphism

The input to the decision problem Graph Isomorphism is two graphs $G_1$ and $G_2$. The problem is to determine if there is an isomorphism between them. That is, a bijection $f$ from $V(G_1)$ to $V(G_2)$ such that $xy$ is an edge of $G_1$ if and only if $f(x)f(y)$ is an edge of $G_2$. This problem is neither known to be in $P$ nor known to be $NP$-complete.

In a probabilistic analysis of Graph Isomorphism, we do not want to consider an input consisting of two random graphs, as they will **whp** be obviously non-isomorphic because, e.g., they have a different number of edges or different degree sequences. There are (at least) two ways of dealing with this problem. The first is to assume that that the input consists of a graph $G$ drawn from the uniform distribution on the $n$ vertex graphs and a second graph $H$ about which we have no information (the reader may wish to think of $H$ as chosen by an adversary who has seen $G$). The second (more studied) approach is to consider canonical labelling algorithms. A canonical labelling algorithm assigns to a graph $G$ on vertex set $\{1, ..n\}$, a permutation $\Pi_G$ such that if two graphs $G$ and $H$ are isomorphic then $\Pi_H^{-1}\Pi_G$ is an isomorphism from $G$ to $H$. That is, a canonical labeling algorithm relabels graphs so that if the original graphs were isomorphic then the relabelled graphs coincide.

As an example, a canonical labelling algorithm might choose to order the vertices of the graph so that if $\Pi(i) < \Pi(j)$ then $i$ is in more triangles than $j$. We note that if no two vertices of $G$ are in the same number of triangles than there is a unique $\Pi_G$ satisfying this condition. Furthermore, if $H$ is isomorphic to $G$ then there is a unique $\Pi_H$ satisfying this condition and $\Pi_G(G)$ and $\Pi_H(H)$ are the same graph. Of course our canonical labelling algorithm must also have a way of dealing with graphs in which some pairs of vertices are in the same number of triangles.

We invite the reader to show that there is a canonical labelling algorithm that runs in $O(n^3 2^n)$ time. We also discuss canonical labelling algorithms which relabel some but not all graphs. In this case, if the algorithm relabels $G$ it should also relabel all graphs isomorphic to $G$.

In this section, we prove a result of Babai, Erdos, and Selkow [**?**] ( for strengthenings see Karp[**?**] and Lipton[**?**]).

**Theorem 2** *There is a canonical labelling algorithm which labels* $G_{n,\frac{1}{2}}$ **whp**.

One such canonical labelling algorithm is to order the vertices in non-increasing order of degree and to order the vertices of the same degree so that vertices in more triangles come first. We shall not treat this algorithm here (however, the reader is invited to show that it succeeds **whp** by showing that the expected number of pairs of vertices with the same degree and in the same number of triangles is $o(1)$). Instead, we treat an algorithm which orders the vertices in non-increasing order of degree but chooses the order in the set of vertices of the same degree in a slightly different way.

We need:

**Definition** We call a degree *unique* if there is precisely one vertex with this degree. We call a vertex *solitary* if it has unique degree.

**Lemma 2** **Whp**, *the highest* $\lceil 3\log n \rceil$ *degrees of* $G_{n,\frac{1}{2}}$ *are unique and no two vertices have the same neighbourhood on the* $\lceil 3\log n \rceil$ *vertices of highest degree.*

Now, the canonical labelling algorithm we consider orders vertices of the same degree so that if $\pi(i) < \pi(j)$ then the highest degree vertex which sees exactly one of $\{i, j\}$ sees $i$ but not $j$. Lemma 2 ensures that this algorithm succeeds $\mathbf{whp}$. Thus the lemma implies the theorem. We prove the lemma below.

**Research Problem:**

**Proof of Lemma 2** Let $l = \lceil 3\log n \rceil$. The key to proving the lemma is to show:

**6** **Whp** *the* $l+1$ *highest degrees in* $G_{n,\frac{1}{2}}$ *are unique and the difference between two consecutive degrees is at least five*

We prove this result below. Combining it with the following result proves the lemma.

**7** *The probability that the* $l+1$ *highest degrees in* $G_{n,\frac{1}{2}}$ *are unique and differ by at least five and two vertices have the same neighbourhood on the* $l$ *vertices of highest degree is* $o(1)$.

To prove (7), we compute the expected number of sets $w_1, ..., w_l, v_1, v_2$ in $G_{n,\frac{1}{2}}$ such that (i) $w_1, ..., w_l$ are solitary vertices with the highest degrees, the $l+1$ highest degrees all differ by at

least five, and (ii) $v_1$ and $v_2$ have the same neighbourhood on $W = \{w_1, ..., w_l\}$. We show that the expected number of such sets is $o(1)$ hence the probability one exists is $o(1)$ and (7) holds.

Now, there are $\binom{n}{l}\binom{n-l}{2}$ choices for $W, v_1, v_2$. For each choice, we determine the edges of $G_1 = G_{n,\frac{1}{2}} - v_1 - v_2$. That is, we take a copy of $G_{n-2,\frac{1}{2}}$ with vertex set $V - v_1 - v_2$. If the $l$ vertices of highest degree in $G_1$ are not distinct then (i) cannot hold, for adding $v_1$ and $v_2$ changes each degree by at most two and the difference between two degrees by at most four. If the $l$ vertices of highest degree in this graph are unique, then for (i) to hold the vertices with these degrees must be those in $W$ which by symmetry occurs with probability $\binom{n}{l}^{-1}$. Given that $W$ is the set of high degree vertices in this graph we see, by considering the edges from $v_1$ and $v_2$, that the probability that (ii) holds is $2^{-l} \leq \frac{1}{n^3}$. Thus, the expected number of $W, v_1, v_2$ such that (i) and (ii) holds is $\binom{n}{l}\binom{n-l}{2}(\binom{n}{l}^{-1}n^{-3}) = o(1)$. So, (7) holds as claimed, we turn now to (6).

To prove (6), we consider the $t(= t(n))$ defined in our discussion of edge-colouring. As promised in that discussion, we will show that **whp**, $G_{n,\frac{1}{2}}$ has a vertex of degree greater than $t$. In fact, we will prove that **whp** it has at least $l + 1$ such vertices, which combined with (5) for j=1, proves (6). We actually prove a much stronger result which we will need later, to wit:

**8** *The probability that there are fewer than $n^{\frac{1}{10}}$ vertices of degree greater than $t$ is $O(2^{\frac{-n}{10}})$.*

To prove this result, we use "the method of deferred decisions" as described in Knuth, Motwani and Pittel [67]. Imagine that we have an assistant and when we want to know whether an edge $uv$ exists, he flips a fair coin and if it comes down heads the edge exists, otherwise it does not. We only do this at most once for each possible pair $u, v$. The order in which we flip the edges is as described in the following procedure.

(1) Set $i = 1$, choose some vertex $v_1$. determine which edges incident to $v_1$ are present.

(2) If $i = n - 1$ stop, otherwise choose the vertex $v_{i+1}$ in $V - v_1, ..., v_i$ which has the most neighbours in $V_i = \{v_1, .., v_i\}$ and determine which edges between $v_{i+1}$ and $V - V_i - v_{i+1}$ are present.

(3) Increment $i$ and return to Step 2.

By analyzing this procedure, we can show:

**9** *The probability that there is some $i < \frac{n}{4}$ such that $v_{i+1}$ has fewer than $\frac{i}{2} - \sqrt{n}$ neighbours in $V_i$ is $O(2^{\frac{-n}{10}})$.*

**Proof** By our choice of $v_{i+1}$, if this occurs, then there are fewer than $\frac{i(n-i)}{2} - (n-i)\sqrt{n}$ edges between $V_i$ and $V - V_i$. However, we expect $\frac{i(n-i)}{2}$ edges between the two sets. Using the

Chernoff bounds, it is easy to show that expected number of sets $S$ of $i < \frac{n}{4}$ vertices such that there are fewer than $\frac{i(n-i)}{2} - (n-i)\sqrt{n}$ edges between $S$ and $V - S$ is $O(2^{\frac{-n}{10}})$ ( we leave the details to the interested reader). The result follows. □

**10** *The probability that there are fewer than $n^{\frac{1}{10}}$ values of $i$ which are less than $\frac{n}{4}$ such that $v_{i+1}$ has more than $\frac{n-i}{2} + (t - \frac{n}{2} + \sqrt{n})$ neighbours in $V - V_i - v_{i+1}$ is $O(2^{\frac{-n}{10}})$.*

**Proof** Now, in the first $i$ iterations, we flip coins only for edges from $V_i$. Thus, after we choose $v_{i_1}$, the coins for the edges from $v_{i_1}$ to $V - V_i - v_{i_1}$ are yet to be flipped, and in fact are those flipped in the next iteration. It follows via the Chernoff bounds, that for $i < \frac{n}{4}$, the probability of the event $E_i$ that $v_{i+1}$ has more than $\frac{n-i}{2} + (t - \frac{n}{2} + \sqrt{n})$ neighbours in $V - V_i - v_{i+1}$ is close to $\frac{1}{n^{\frac{4}{5}}}$ and is certainly greater than $n^{\frac{-5}{6}}$. Thus, the expected number of $i$ for which $E_i$ holds is at least $n^{\frac{1}{6}}$. Furthermore, for distinct $i$ and $j$, $E_i$ and $E_j$ are independent for they are determined by disjoint sets of edges (the coins for which are flipped in different iterations of our procedure for generating $G_{n,\frac{1}{2}}$). Thus, by applying, e.g., the Chernoff Bounds, we obtain that the number of $i$ for which $E_i$ holds is less than $\frac{n^{\frac{1}{6}}}{2}$ with a probability which is $o(2^{\frac{-n}{10}})$. □

Combining (9) and (10) yields (8) thereby completing the proof of the lemma. □

We close this section by remarking that combining (5) and (8) yields the following result, which we shall find useful:

**11** *The probability that there are fewer than $\frac{n^{\frac{1}{10}}}{4}$ solitary vertices of $G$ with degree bigger than $t$ is $O(2^{-\frac{n}{10}})$.*

## 2.2 Polynomial Expected Time

### 2.2.1 Graph Isomorphism

We now present a polynomial expected time algorithm for graph isomorphism. uniform distribution on $n$-vertex graphs and a graph $H$ about which we have no information.

As a last resort, our algorithm uses the brute force $O(n^2 n!)$ procedure of testing each of the $n!$ bijections between $V(G)$ and $V(H)$.

Our algorithm also uses two sub-algorithms both of which are reminiscent of the canonical labelling procedure in the last section. In the canonical labelling procedure, we essentially knew the bijection on some subset $S$ of $V$ (the high degree solitary vertices) and this allowed us to determine the rest of the bijection, simply by considering $N(v) \cap S$ for each $v \in V - S$.

To ease our discussion of extending partial bijections in this manner, we need some definitions. Let $S \subseteq V(G)$, we say a vertex $v$ in $V - S$ is *determined* by $S$ if there is no $w \in V - S$ with $N(v) \cap S = N(w) \cap S$. We let $det(S)$ be the set of vertices determined by $S$. We note:

**Lemma 3** *If $S \subseteq V(G)$ and $f$ is a bijection from $S$ to some subset of $V(H)$, then for any isomorphism $f'$ extending $f$ and for any $v \in det(S)$, we have only one candidate for $f'(v)$ and in $O(n^2)$ time, we can either*

*(i) determine that there is no isomorphism from $G$ to $H$ extending $f$, or*

*(ii) find a bijection $g$ from $det(S) \cup S$ to a subset of $V(H)$ such that any isomorphism $f'$ extending $f$ corresponds with $g$ on $det(S) \cup S$.*

**Proof** We leave this as an exercise for the reader. □

We need to take this idea one step further. To this end, we say a vertex $v$ in $V - S$ is *fixed* by $S$ if $v \in det(S) \cup det(det(S))$. We let $fix(S)$ be the set of vertices fixed by $S$. Applying Lemma 3 twice, we obtain:

**Lemma 4** *If $S \subseteq V(G)$ and $f$ is a bijection from $S$ to some subset of $V(H)$, then for any isomorphism $f'$ extending $f$ and for any $v \in fix(S)$, we have only one candidate for $f'(v)$ and in $O(n^2)$ time, we can either*

*(i) determine that there is no isomorphism from $G$ to $H$ extending $f$, or*

*(ii) find a bijection $g$ from $fix(S) \cup S$ to a subset of $V(H)$ such that any isomorphism $f'$ extending $f$ corresponds with $g$ on $fix(S) \cup S$.*

The probabilistic results we need are:

**Lemma 5** *With probability $1 - O(2^{-n^{\frac{1}{10}}})$, the solitary vertices fix $V$.*

**Lemma 6** *With probability $1 - O(2^{-4n \log n})$, every set $S$ of $\lceil 20 \log\ n \rceil$ vertices fixes all but at most $\lceil 20 \log\ n \rceil$ vertices of $G$.*

We prove these results in a moment. First, we show that they imply the existence of the desired polynomial expected time algorithm.

We will use an algorithm $A_1$ which computes the degree sequence of $G$ and $H$, ensures that these coincide, sets $S$ to be the set of solitary vertices of $G$, sets $S'$ to be the set of solitary vertices of $H$, and lets $f$ be the bijection from $S$ to $S'$ such that $d_G(v) = d_H(f(v))$. It then determines if $S$ fixes $V(G)$. If not it halts. Otherwise, applying the algorithm of Lemma 4, it either determines and outputs that $G$ is not isomorphic to $H$ or extends $f$ to a bijection $g$ from $V(G)$ to $V(H)$ such that the only possible isomorphism from $G$ to $H$ is $g$. If it returns such a bijection $g$, it then checks whether or not $g$ is in fact an isomorphism. If so, it outputs this isomorphism, otherwise it outputs the fact that $G$ and $H$ are not isomorphic. By Lemma 4, an answer returned by the

algorithm is correct. By Lemma 5, the probability that $A_1$ does not give an answer is $O(2^{-n^{\frac{1}{10}}})$. It is straightforward to verify that the algorithm can be implemented in $O(n^2)$ time.

We will also use an algorithm $A_2$ which first chooses an arbitrary set $S$ of $\lceil 20 \log\ n \rceil$ vertices of $G$. The algorithm then checks if $S$ fixes all but at most $\lceil 20 \log\ n \rceil$ vertices of $G$. If not it halts. The algorithm next determines for each set $S'$ of $|S|$ vertices of $H$ and bijection $f$ from $S$ to $S'$ whether or not there is isomorphism extending $f$. If it finds for some $S'$ and $f$ that there is an isomorphism extending $f$, it returns with the information that $G$ and $H$ are isomorphic. If it determines that for each $S'$ and $f$ there is no isomorphism extending $f$ then it outputs that $G$ and $H$ are not isomorphic.

For a given $S'$ and $f$, applying the procedure of Lemma 4, $A_2$ either determines and outputs that no isomorphism from $G$ to $H$ extends $f$ or extends $f$ to a bijection $g$ from $fix(S) \cup S$ to a subset of $V(H)$ such that the only possible isomorphisms from $G$ to $H$ extending $f$ also extend $g$. If it returns such a bijection $g$, it then checks whether or not any of the at most $|V - fix(S) - S|! \leq \lceil 20 \log\ n \rceil!$ extensions of $g$ to bijections from $V(G)$ to $V(H)$ are isomorphisms. If any of these are isomorphisms, the algorithm returns that there is an isomorphism extending $f$, otherwise it returns that no such isomorphism exists. By Lemma 4, an answer returned by the algorithm is correct. By Lemma 6, the probability that $A_2$ does not give an answer is $O(2^{-4n \log\ n})$. It is straightforward to show that the algorithm can be implemented so that it spends $O(n^2 \lceil 20 \log\ n \rceil!)$ time on each pair $(S', f)$ and hence takes at most $O(n^{\lceil 20 \log\ n \rceil} n^2 \lceil 20 \log\ n \rceil!) = o(n^{60 \log\ n})$ time in total.

Now, our global algorithm applies $A_1$, then applies $A_2$ if $A_1$ terminates without a response, and finally applies our brute force algorithm if $A_2$ fails to provide an answer. By the above remarks, the expected running time of this algorithm is $O(n^2) + 0(2^{-n^{\frac{1}{10}}} n^{60 \log\ n}) + O(2^{-4n \log n} n^2 n!) = O(n^2)$. Since a random graph has $O(n^2)$ edges clearly this algorithm has optimal expected running time. We can actually create a canonical labelling algorithm whose expected running time is $O(n^2)$ using similar techniques, see Babai and Kucera[**?**] for a result in this vein.

With our description of the algorithm complete, it remains only to prove our two probabilistic lemmas

We need the following auxiliary results, all of which can be proven using simple First Moment calculations:

**12** *The probability that there is a set $S$ of $\lceil 20 \log n \rceil$ vertices which determines fewer than $\frac{2n}{3}$ vertices is $O(2^{-4n \log n})$.*

**13** *The probability that there is a set $S$ of $\frac{2n}{3}$ vertices which determines fewer than $\frac{n}{3} - 20 \log n$ vertices is $O(2^{-4n \log n})$.*

**14** *The probability that there is a set $S$ of $\frac{2n}{3}$ vertices which does not determine $V - S$ is $o(2^{\frac{-n}{10}})$.*

Now, Lemma 6 follows from (12) and (13). Lemma 5 follows from (12) and (14), and (11).

### 2.2.2 Hamilton Cycles

We now present an algorithm DENSEHAM for Hamilton Cycle that has expected running time which is $O(n^5)$. The algorithm uses two sub-algorithms. One, $A_2$, solves Hamilton cycle on any graph in $O(n^3 2^n)$ time and actually finds the cycle if it exists. It is the Dynamic Programming algorithm of Held and Karp [55]. The other, $A_1$ runs in $O(n^4)$ time. It attempts to construct a Hamilton cycle in the input graph. The probability that it fails to return a Hamilton cycle when applied to $G_{n,\frac{1}{2}}$ is $O(2^{-n}n^2)$. DENSEHAM first applies $A_1$ and then applies $A_2$ if $A_1$ fails to find a Hamilton cycle. Clearly, DENSEHAM does indeed solve Hamilton Cycle, and in fact outputs a Hamilton cycle if one exists. Furthermore, its expected running time is $O(n^4) + O(2^{-n}n^2)O(2^n n^3) = O(n^5)$, as claimed. It remains only to describe and analyse $A_1$ and $A_2$.

$A_2$ is a simple dynamic programming algorithm which determines for each subset $S$ of $V$ with $|S| \geq 2$, and for each pair of vertices $\{u, v\}$ of $S$, whether or not there is a Hamilton path through $S$ with endpoints $u$ and $v$. To determine if $G$ has a Hamilton cycle we need then only check if for any edge $uv$ of $G$ there is a Hamilton path through $S = V$ with endpoints $u$ and $v$. $A_2$ considers the subsets of $V$ in increasing order of size. To determine if there is a Hamilton path of $S$ with endpoints $u$ and $v$, it simply checks whether there is some neighbour $v'$ of $v$ in $S$ such that there is a Hamilton path of $S - v$ with endpoints $u$ and $v'$. Since the algorithm has already considered $S - v$, this can be done via a simple table lookup. We spend $O(n)$ time on each triple $S, u, v$ so the the claimed running time bound on $A_2$ holds. With a little extra bookkeeping we can also construct the Hamilton cycle, we omit the details.

$A_1$ is reminiscent of the algorithm for Hamilton Cycle presented in the last section. We show:

**Lemma 7** *Let $G$ be a sufficiently large graph such that*

*(i) $\exists$ a set $S$ of at most 5000 vertices such that $G - S$ is tractable,*

*(ii) the minimum degree of $G$ is at least 2, and*

*(iii) at most one vertex of $G$ has degree less than 15000.*

*Then $G$ has a Hamilton cycle. Furthermore, given $S$ we can find the Hamilton cycle in $O(n^4)$ time.*

We also show that the probability that $G_{n,\frac{1}{2}}$ satisfies conditions (i)-(iii) of Lemma 7 is $O(\frac{n^2}{2^n})$. Actually we prove a slightly stronger result which permits us to use a greedy procedure for finding $S$.

**Definition** A *bad sequence of length $l$* is a sequence $\{X_1, ..., X_l\}$ of disjoint subsets of $G$ such that letting $G^i = G - \cup_{j \leq i} X_j$, we have that for each $i$ between 0 and $l - 1$, either

(a) $X_{i+1}$ is a vertex $v$ such that $d_{G^i}(v) - \frac{|V(G^i)|}{2} > \frac{|V(G^i)|}{50}$,

(b) $X_{i+1}$ is a pair $u, v$ such that $|N_{G^i}(u) \cup N_{G^i}(v)| - \frac{3|V(G^i)|}{4} > \frac{|V(G^i)|}{50}$,

(c) $X_{i+1}$ is a triple $u, v, w$ such that $|N_{G^i}(u) \cup N_{G^i}(v) \cup N_{G^i}(w)| - \frac{7|V(G^i)|}{8} > \frac{|V(G^i)|}{50}$,

**Lemma 8** *With probability* $1 - O(\frac{n^2}{2^n})$, $G_{n,\frac{1}{2}}$ *has minimum degree 2, has at most one vertex of degree less than 15000, and has no bad sequence of length 1500.*

Now, algorithm $A_1$ proceeds as follows. It firsts ensures that $G$ has maximum degree at least two and at most one vertex of degree less than 15000. If this is not true, the algorithm terminates with no output. Otherwise, it generates a maximal bad sequence $\{X_1, ..., X_l\}$ of length at most 1500 (i.e. the sequence either has length 1500 or cannot be extended). This can be done in $O(n^4)$ time because having found $\{X_1, ..., X_i\}$ we can search for $X_{i+1}$ simply by checking whether any of the $\binom{n}{3} + \binom{n}{2} + n$ sets of size at most 3 in $G$ satisfy one of conditions (a)-(c) in the definition of bad sequence. If the bad sequence $A_1$ finds has length 1500, it terminates without output. Otherwise, it sets $S = \cup_{i=1}^{l} X_i$, and applies the algorithm of Lemma 7 to construct a Hamilton cycle in $G$ in $O(n^4)$ time (we note that G-S is tractable by the maximality of the bad sequence). By Lemma 8, the probability that $A_1$ fails to return a Hamilton cycle is $O(\frac{n^2}{2^n})$ as claimed. This completes our description of $A_1$ and $DENSEHAM$, it remains only to prove the two lemmas.

**Proof of Lemma 8** The probability that a vertex $v$ of $G_{n,\frac{1}{2}}$ has degree 0 or 1 is $\frac{n+1}{2^n}$. Thus, the probability that the minimum degree of $G_{n,\frac{1}{2}}$ is 0 or 1 is $O(\frac{n^2}{2^n})$. The probability that there are two vertices of $G_{n,\frac{1}{2}}$ of degree less than 15000 is $O(\binom{n}{2}(\frac{n^{15000}}{2^n})^2) = o(2^{-n})$.

Finally, the probability that some $\{X_1, ..., X_{1500}\}$ is a bad sequence is, via an application of the Chernoff bounds, $O((2^{\frac{-n}{1250}})^{1500})$. Hence, the expected number of bad sequences of length 1500 is $o(2^{-n})$. The result follows. □

**Proof of Lemma 7** The key to the proof is the following auxiliary result.

**15** *Let $H$ be a graph which is the union of a tractable graph $G$ and a matching $M \subset G$ with fewer than 5000 edges. Then provided $H$ is sufficiently large it has a Hamilton cycle $C$ such that $M \subseteq E(C)$. Furthermore, we can find such a Hamilton cycle in $O(n^4)$ time.*

**Proof** The first step in the proof of (15) is to find a path $Q$ in $H$ with $M \subseteq E(Q)$ and such that $Q$ has at most $3|M|$ edges. This can be done greedily because every two vertices of $G$ have more than $\frac{n}{5}$ common neighbours. We then apply Phases 1-3 of the algorithm for constructing a Hamilton cycle presented in the last section initializing with $P = Q$, and ensuring that we never delete an edge of $Q$ from the path or cycle we create ( this is possible because $Q$ has only a bounded number of edges; we note that in Phase 2 we will let $w$ be an endpoint of $P$ which is not in $Q$). □

We turn now to the proof of Lemma 7. We enumerate $S$ as $s_1, ..., s_k$ (with $k < 5000$) so that $s_1$ is the lowest degree vertex of $S$. We first consider the case in which $s_1$ has exactly one

neighbour $x$ in $V - S$. In this case, we know that $s_1$ must have a neighbour in $S$, wlog $s_2$. Since for $i > 1$, $s_i$ has at least 15000 neighbours, we can find distinct vertices $x_2, ..., x_l, y_2, ..., y_l$ of $V - S$ such that for $i \geq 3$, $s_i x_i, s_i y_i \in E(G)$,$x_2 = x$, and $s_2 y_2 \in E(G)$. We set $M = \{x_2 y_2, ..., x_l y_l\}$, and apply the algorithm of (15) to $H = (G - S) \cup M$. We let $C$ be the output Hamilton cycle in $H$ with $M \subseteq E(H)$. We let $C'$ be the Hamilton cycle in $G$ with edge set $E(H) - M \cup (\cup_{i=3}^{l} \{x_i s_i, s_i y_i\}) \cup \{x s_1, s_1 s_2, s_2 y_2\}$.

The cases in which $s_1$ has 0 or more than 2 neighbours in $V - S$ are similar, we omit the details. □

**Exercise:** Combine this algorithm with our earlier algorithm to develop an algorithm for Hamilton cycle whose expected running time on $G_{n,\frac{1}{2}}$ runs in $O(n^2)$ time (and hence is linear in the size of the input).

The first polynomial expected time algorithms were developed independently by Bollobás, Fenner and Frieze [13] and by Gurevich and Shelah [53].

## 2.3 Edge Colouring

Perkovic and Reed[**?**] recently developed a polynomial expected time algorithm for edge colouring. Their algorithm is much too complicated to explain in detail here. The complexity is due to the fact that the fastest known edge colouring algorithm which succeeds on all graphs has a worst-case running time bound which is $O(2^{cn^2})$ on $n$ vertex graphs for some $c > 0$. We will briefly outline their algorithm, to do so we need a few auxiliary results.

We use $\Delta(G)$ for the maximum degree in $G$.

**Definition 1** *$H$ is an $l$-reduction of $G$ if $\Delta(H) = \Delta(G) - l$ and there exist matchings $M_1, ..., M_l$ in $G$ such that $H = G - \cup_{i=1}^{l} M_i$. $H$ is a reduction of $G$ if it is an $l$-reduction for some $l$.*

**Observation 1** *If a reduction $H$ of $G$ has a $\Delta(H)$ edge colouring then $G$ has a $\Delta(G)$ edge colouring.*

**Definition 2** *A subgraph $H$ of $G$ is over-full if $|V(H)|$ is odd and $|E(H)| > \Delta(G)\frac{|V(H)|-1}{2}$.*

**Observation 2** *If $G$ contains an over-full subgraph then it has no $\Delta$ edge colouring.*

**Proof** If $H$ has $2k + 1$ edges then the largest matching in $H$ has $k$ edges. □

**Theorem 3 Padberg and Rao[?]** *There is a polynomial time algorithm which determines if $G$ has an over-full subgraph.*

**Theorem 4 [44]** *The probability that $G_{n,\frac{1}{2}}$ has a reduction $H$ whose vertices of maximum degree form a stable set is $1 - O(n^{c_1 n})$ for some $c_1 > 0$. Furthermore, there is a polynomial time algorithm which finds such a reduction and corresponding matchings $M_1, ..., M_l$ with this probability.*

**Corollary 2** *There is an polynomial time algorithm which $\Delta$ edge colours $H$ with probability $1 - O(n^{c_1 n})$ for some $c_1 > 0$.*

**Proof** We attempt to find a reduction $H$ of $G$ whose vertices form a stable set using the algorithm of the theorem. If we succeed, we apply Berge and Fournier's algorithm to edge colour $H$ and then use the matchings $M_1, ..., M_l$ to colour the remaining edges of $G$. □

**Theorem 5 [44]** *There exists a $c_2 > 0$ such that for $n > 3$, the probability that $G_{n,\frac{1}{2}}$ has an over-full subgraph is at mosst $n^{-c_2 n}$.*

**Definition 3** *A graph is bipartite if it can be partitioned into two stable sets. A graph $G$ is near-bipartite if for some vertex $v$, $G - v$ is bipartite.*

**Theorem 6 [?]** *A near bipartite graph $G$ is $\Delta$ edge colourable if and only if it contains no over-full subgraph. Furthermore, there is a polynomial time algorithm which given a near-bipartite graph either finds an over-full subgraph or a $\Delta$ edge colouring.*

Perkovic and Reed's algorithm first applies the polynomial time algorithm of Corollary 2 which fails with probability $O(n^{-c_1 n}$ for some constant $c_1$. They then apply the algorithm of Theorem 6 to colour any remaining over-full subgraph. There are two more algorithms which might be applied. The first $Cleanup_1$ runs in $O(2^n)$ time and attempts to find a $\Delta$ edge colouring of a graph with no over-full subgraph. It fails with probability $O(2^{-cn^2})$ for some $c$. The second $Cleanup_2$ is a dynamic programming algorithm which optimally colours every graph and has running time which is smaller than the inverse of the probability that $Cleanup_1$ fails. It follows that applying the four algorithms in the given order yields a polynomial expected time algorithm. We omit the description of $Cleanup_2$. $Cleanup_1$ more or less finds a near-bipartite reduction $H$ of $G$, and applies the algorithm of Theorem 6 to find a $\Delta(H)$ edge colouring of $H$. Actually, the algorithm finds a reduction of a graph which is derived from $H$ and may have multiple edges. We omit any further description.

## 2.4 Further Results

**Hamilton Cycles for Sparse Graphs** As we have seen, finding a Hamiltonian cycle in a dense graph is relatively easy. The analysis for sparse graphs is more intricate but still based on the two

procedures used in Phase 1 of our algorithm for tractable graphs. That is, *extension* of the path by adding a neighbour of an endpoint, and *rotation* of the path $P = vP'yP''$ to obtain $P'vyP''$. By iteratively applying rotations before extending, Bollobás, Fenner and Frieze [13] develop a polynomial time algorithm $HAM$ with the property that for all $m = m(n)$

$$\lim_{n\to\infty} \mathbf{Pr}(\text{HAM finds a Hamilton cycle}) = \lim_{n\to\infty} \mathbf{Pr}(G_{n,m} \text{ is Hamiltonian}).$$

Frieze [41] proved a similar result for random digraphs.

**Research Problem** Develop an algorithm which runs in polynomial expected time on $G_{n,m}$ for every $m$.

**Graph Colouring** As we shall see in Section **??**, there is no known polynomial time algorithm which optimally vertex colours $G_{n,\frac{1}{2}}$ with high probability. There has been some success in designing algorithms that **whp** optimally vertex colour randomly generated $k$-colourable graphs, for small $k$. The strongest current results stem from the spectral approach of Alon and Kahale [5]. Chen and Frieze [25] used this approach to colour random hypergraphs. The $k$-colouring algorithm of Dyer and Frieze [32] optimally colours in polynomial expected time.

**Min Bisection** We are given a graph $G$ and asked to divide the vertices into two sets of equal size so as to minimise the number of edges between them. Most analysis has been concerned with the case where there is a fixed planted bisection with many fewer edges than expected. Bui, Chaudhuri, Leighton and Sipser [22] considered random regular graphs and showed how to find the planted cut in polynomial time **whp**. Dyer and Frieze [32] did the same for $G_{n,p}$, $p$ constant. The strongest results on this problem have been obtained by Boppana [14] using spectral techniques. Jerrum and Sorkin [57] analysed a version of simulated annealing on $G_{n,m}$.

# 3 Faster algorithms for easy problems

In this section, we discuss the probabilistic analysis of algorithms for which polynomial time algorithms are known to exist. Typically, we analyze a simple algorithm for the problem and show that its expected running time is much better than its worst case running time. Our three representative examples, shortest paths, matchings, and linear programming, are the foundations on which the field of combinatorial optimization is built.

## 3.1 Perfect Matchings

Recall that a matching is a set of edges no two of which are incident. A vertex $v$ is *covered* by a matching $M$ if it is in an edge of $M$, otherwise it is *uncovered*. A matching is *perfect* if it covers all the vertices. The fastest algorithm for determining if a graph with $n$ vertices and $m$ edges has a perfect matching has a worst case running time of $O(n^{1/2}m)$. In this section we

show that an appropriate implementation of this algorithm runs in linear expected time on $G_{n,1/2}$, $n$ even. There are two phases. Phase 1 *greedily* chooses edges and finds a matching of size $n/2 - O(\log n)$ **whp**. Phase 2 uses augmenting paths of length 3 (that is repeatedly replaces an edge $xy$ of the matching by two edges $wx$ and $yz$ where $w$ and $z$ were previously uncovered) to produce a perfect matching **whp**.

Recall that $V(G_{n,\frac{1}{2}}) = \{1, .., n\}$.

**Phase 1**
In this procedure $S$ will denote the vertices not covered by the matching $M$ produced so far.

In iteration $i$, we choose the minimum $x_i$ of $S$ and find the smallest numbered vertex $y_i$ it can be matched to (i.e. the smallest $y_i$ which is still uncovered and is adjacent to $x_i$). If there is no such $y_i \in S$ we terminate Phase 1, else we add $x_i y_i$ to $M$ and repeat.

Suppose Phase 1 produces $M = \{x_1y_1, x_2y_2, \ldots, x_py_p\}$ and that $M$ leaves $Z = \{z_1, z_2, \ldots, z_{2q}\}$, $q = \frac{1}{2}n - p$ unmatched. Note that for each $i$, $x_i < y_i$. We set $X = \{x_1, ..., x_p\}$. We set $z^* = min\ Z$.

**Phase 2**
In this phase we take the members of $Z$ in pairs $z_{2i-1}, z_{2i}$, $i = 1, 2, \ldots q$ and try to find $x_t y_t$ such that $z_{2i-1}x_t$ and $z_{2i}y_t$ are both edges. In which case we delete edge $x_ty_t$ from $M$ and add the edges $z_{2i-1}x_t, z_{2i}y_t$. For each $i$ we go sequentially through values of $t$, starting the $i$th search at the beginning of the path. If we fail for some $i$ then the whole algorithm fails.

We now discuss the probability that we fail to find a perfect matching in $G_{n,1/2}$ this way. Our analysis fits the notion of "the method of deferred decisions" described in Section 2.1.3.

First consider Phase 1. We claim that in this phase we need only examine the presence of each edge once. To see this note that in iteration $i$, we only examine edges from $x_i$ to $S - x_i$. But any edge examined in a previous iteration has an endpoint $x_j$ with $j < i$ and $x_j$ is no longer in $S$, the claim follows. Furthermore, if we flip the coin for an edge $uv$ incident to some vertex $v$ in this iteration and find it exists then we add $uv$ to $M$ and will flip no more coins for edges incident to $v$ in this Phase. Thus if we test for the presence of $t$ edges incident to $v$ and find none of them exist then these must be the first $t$ edges incident to $v$ examined, and so this occurs with probability $\left(\frac{1}{2}\right)^t$. For $\xi \in S \cup X$ and $K > 0$ we define the event

$$\mathcal{E}_\xi = \{|\{j | x_j < \xi < y_j\}| \geq Kn \log_2 n\}.$$

Then we have:

1. $\mathbf{Pr}\left(\bigcup_{\xi \in S \cup X} \mathcal{E}_\xi\right) \leq n^{1-K}$.

**Proof** For each such $j$, we failed to find the edge $x_j\xi$. □

2. $\mathbf{Pr}(\exists i \leq p :\ y_i - x_i \geq 2K \log_2 n) \leq 2n^{1-K}$.

**Proof** For each such $i$, either $\mathcal{E}_{x_i}$ occurs or the first $K \log_2 n$ edges examined in the $i$th iteration are not present. □

3. $\mathbf{Pr}(\min Z \leq n - 2K \log_2 n) \leq 2n^{1-K}$.

**Proof** If this occurs then either $\mathcal{E}_z$ occurs for the minimum $z$ in $Z$ or the first $K \log_2 n$ edges examined in the final iteration are not present. □

Assume next that none of the events described in 1,2,3 above occur and consider Phase 2. We observe that for any edge $x_i y_i$ of $M$ we have not flipped the coin for the edges $x_i k, y_i k$ for $k > y_i$, so if $y_i < z^*$ we have not flipped the coin for $x_i z$ or $y_i z$ for any $z \in Z$. Since $x_i < 2i$, it follows from 2 and 3 that we have not flipped the coins for $x_t z$ or $y_t z$ where $z \in Z$ and $t \leq n/3$. So when we search for an alternating path of length 3 for the pair $z_1, z_2$, the probability that we need $2K \log_2 n$ attempts is $\left(\frac{1}{4}\right)^{2K \log_2 n} = n^{-K}$. The $i$th pair examines edges with endpoints $z_{2i-1}, z_{2i}$ which have not been examined in previous Phase 2 iterations and so we see in a similar way, that Phase 2 fails with (conditional) probability at most $n^{-K} K \log_2 n$.

In summary, this algorithm finds a perfect matching with probability at least $1 - O(n^{1-K})$ after flipping at most $2Kn \log_2 n$ coins.

## 3.2 Linear Programming

It was observed early on that the simplex algorithm and its variants worked remarkably well in practise. A theoretical explanation was sought for this through probabilistic analysis, especially as Klee and Minty [66] had shown that a standard variant did not run in worst-case polynomial time.

The first average-case results were due to Borgwardt [15] and Smales [82, 83]. The model chosen in [15] is not the most obvious and [82, 83] requires that the number of constraints be small. Blair [10] later gave a simplified explanation for the results of [82, 83] — see Section 3.2.1. Further work on this problem came through another change of probabilistic model where randomness is introduced through a random choice of $\leq$ or $\geq$ for a particular constraint. See Haimovich [54], Adler and Megiddo [2], Adler, Karp and Shamir [1] and Adler, Megiddo and Todd [3]. A recent book by Borgwardt [16] covers this subject in detail.

There are still unanswered questions in this area: For example, can one find a reasonable model plus a proof that the algorithm which always chooses a variable of largest reduced cost to enter the basis runs in polynomial expected time.

### 3.2.1 Blair's Analysis

In this section we prove a simple result based on the ideas of Blair [10]. The result given here is not as strong but has a much simpler analysis.

In Blair's model we have a linear program

$$\begin{array}{lllll} \text{Maximise} & cx & & \\ \text{Subject to} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

Here $A$ is an $(m-1) \times n$ matrix.

We use the following notation: for a matrix $M$, $M_{(i)}$ denotes its $i$th row and $M^{(j)}$ denotes its $j$th column.

It is assumed that $b$ is non-positive but arbitrary (x=0 is a feasible solution) and $A, c$ are produced as follows: let $\hat{A} = \left[\begin{smallmatrix} c \\ A \end{smallmatrix}\right]$ have rows indexed by $\{0, 1, \ldots, m-1\}$. We have an $m \times n$ matrix $B$ in which no two elements in the same row are the same. $\hat{A}_{(i)}$ is an independent random permutation of the corresponding row $\hat{B}_{(i)}$.

Column $\hat{A}^{(j)}$ *dominates* column $\hat{A}^{(k)}$ if $\hat{A}(i,j) > \hat{A}(i,k)$ for $i = 0, 1, \ldots, m-1$. It is easy to see that no optimal solution will have $x_k > 0$ if $\hat{A}^{(k)}$ is dominated by some other column.

Several versions of the simplex algorithm have the following property:
No variable corresponding to a dominated column of $\hat{A}$ enters the basis at any iteration.

As examples:

- Try to choose a surplus variable to enter, otherwise choose the entering variable with the largest reduced cost.
- Delete dominated columns at the start.
- The path following algorithm of [82, 83].

So, if we let $L$ be the number of undominated columns of $\hat{A}$, then these algorithms require at most $\binom{L+m-1}{m-1}$ iterations. Below, we sketch a proof that **whp**

$$L \leq m^{3m \log\log n}. \tag{1}$$

In which case,

$$\binom{L+m-1}{m-1} \leq \left(\frac{3L}{m}\right)^m \leq m^{3m^2 \log\log n}$$

So if $m$ is small i.e. $O((\log n)^{1/2}/\log\log n)$ the algorithms take a polynomial number of iterations **whp**.

**Proof of (1)**
We actually prove:

$$\mathbf{E}(L) \leq m^{2m \log\log n}. \tag{2}$$

From which the result follows. Let $\alpha = \left(\frac{\log n}{n}\right)^{1/m}$ Consider $i = 0$ and let $I_k$ be the index set of the $\alpha n$ largest elements of $\hat{A}_{(k)}$. Let $I = \bigcap_{k=0}^{m-1} I_k$. Then

$$\mathbf{E}(|I|) = \alpha_0^m n.$$

**Exercise:** show that $\mathbf{Pr}(|I| = 0) \leq \frac{1}{n}$.

Any column not in $I_0 \cup I_1 \cup \cdots \cup I_{m-1}$ is dominated by a column with index in $I$. So, using the result of the exercise, the expected number of undominated columns exceeds the sum of the number of undominated columns in each $I_i$ by at most 1. Letting $f(m, n)$ be the expected number of undominated columns in a matrix with $n$ columns and $m$ rows each of which is uniformly randomly permuted, we obtain:

$$f(m, n) \leq m f(m, \alpha n) + 1.$$

Checking inductively that $f(m, n) \leq m^{2m \log\log n}$ yields the desired result.

## 3.3 Shortest Paths

Most work in this area has been restricted to that of finding shortest paths between all pairs of nodes in a complete digraph with independently chosen random non-negative edge weights. More generally, one considers distributions which are *endpoint independent*. Loosely, this means that if the edges leaving a vertex are sorted according to their cost, then the associated endpoints occur in random order. Spira [84] showed that using a heap in a version of Dijkstra's algorithm [31] gave a solution in $O(n^2(\log n)^2)$ expected time. This was improved by Bloniarz [11] and Frieze and Grimmett [43]. Moffatt and Takaoka [77] subsequently reduced the expected running time to $O(n^2 \log n)$. Recently, Mehlhorn and Priebe [74] show this algorithm runs in time $O(n^2 \log n)$ **whp** and not just in expectation. They also give an $O(n \log n)$ lower bound for the single source problem under a class of distributions.

Luby and Ragde [71] consider the problem of finding a single shortest path between a source $s$ and a sink $t$. They show that searching simultaneously from *both* $s$ and $t$ can be efficient on average. For example they give a $\Theta(\sqrt{n} \log n)$ time bound assuming sorted edge lists and edge lengths chosen independently from "reasonable" distributions.

**Spira's Algorithm**
For each $v \in V$ we keep a list $L_v$ of the edges $(v, w)$, $w \neq v$ sorted in increasing order of length. It takes $O(n^2 \log n)$ time to produce these lists. By the assumption of endpoint independence these orderings are random and independent of each other. We keep pointers $p_v, v \in V$ which are initialised to point to a dummy element preceding the first real element of $L_v$.

The algorithm consists of $n$ single source shortest path problems, one for each $v \in V$. Consider one such problem for some $s \in V$. As usual the algorithm incrementally produces a set $S$ (initially $S = \{s\}$) containing those vertices $v$ for which a shortest path from $s$ to $v$ has been

calculated. For each $v \in S$ we keep a value $d(v)$. When $v$ is added to $S$ we have

$$d(v) = dist(s, v) + \min_{w \notin S} \ell(v, w). \tag{3}$$

We do not immediately update $d(v)$ each time we update $S$. This saves time on average.

The algorithm needs a subsidiary data structure $Q$ called a *priority queue*.
$Q$ admits the following operations: insert an item, delete an item and determine the item of minimum value. Each such operation takes $O(\log n)$ time.

An iteration of Spira's algorithm consists of

1. (a) Determine the minimum value $d(v) = dist(s, v) + \ell(v, w)$ in $Q$;
   If $w \notin S$ then
   i. Add $w$ to $S$;
   ii. $dist(s, w) := d(v)$;
   iii. goto 2.
   (b) Otherwise: move $p_v$ one position to the next vertex $w'$ on $L_v$;
   (c) Replace $d(v)$ by $dist(s, v) + \ell(wv, w')$ and update $Q$; goto 1
2. Currently $p_w$ is pointing to a dummy element of $L_w$. Move along the list $L_w$ until we find the first $x \notin S$.
3. Put $d(w) = dist(s, w) + \ell(w, x)$ and insert this value into $Q$.

It is straightforward to show that this algorithm solves the all-pairs shortest path problem.

**Time Analysis**
We argue that if $|S| = k$ then the expected number of times we find $w \in S$ in Step 1 and the expected number of updates of $p_w$ needed to find an element not in $S$ in Step 2 is $O(n/(n-k))$. Thus the total expected running time for each single source shortest path problem is of the order

$$\sum_{k=1}^{n-1} \frac{n}{n-k} \log n = O(n(\log n)^2).$$

The bound $O(n/(n-k))$ is explained as follows: Suppose in Step 1 $p_v$ points to the $t$th member of $L_v$. By the endpoint-independent assumption the remaining $n-1-t$ items of $L_v$ are in random order. The first $t$ elements of $L_v$ are in $S$. Thus the probability that the $t+1$th vertex is in $S$ is at most $\frac{k}{n-1}$, conditional on the history of the process so far. The next iteration of Step 1 may involve a different value for $v$, but this probability bound remains true. Thus if $X$ is the random number of moves needed to find a vertex not in $S$. Then

$$\mathbf{Pr}(X > x) \leq \left(\frac{k}{n-1}\right)^x.$$

and

$$\mathbf{E}(X) \leq \sum_{x=1}^{\infty} \left(\frac{k}{n-1}\right)^x = \frac{n-1}{n-k-1}.$$

The same argument suufices for the analysis of Step 2.

The are only a few papers we know of that deal with arbitrary, as opposed to non-negative weights. Kolliopolous and Stein [68] modify the Bellman-Ford dynamic programming algorithm and show that a single source problem can be solved in $O(n^2 \log n)$ expected time when the distribution is endpoint independent. Their model allowed negative cycles. Cooper, Frieze, Mehlhorn and Priebe [29] consider a model in which the arc costs $c_{i,j}$ are generated from

$$c_{i,j} = -u_i + u_j + v_{i,j},$$

where $v_{i,j} \geq 0$. It is assumed that the $v_{i,j}$'s are independent, identically distributed, bounded and their common probability function $F$ satisfies $F'(0) > 0$. The $u_i$'s are arbitrary and of size $O(n/(\log n)^2)$. The algorithm does not see the $u$'s and $v$'s, only the values $c_{i,j}$. They show that a single source shortest path problem can be solved in $O(n^2)$ expected time and an all pairs shortest path problem can be solved in $O(n^2 \log n)$ expected time.

# 4 Asymptotic Optimality and Approximation

In this chapter, we change the focus of our probabilistic analysis. We examine polynomial time algorithms which do not necessarily return optimal solutions and examine how while they perform on typical instances. We discuss Bin Packing, the Euclidean and Asymmetric TSP, and disjoint paths problems.

## 4.1 Bin packing

In its simplest form we are given $x_1, x_2, \ldots, x_n \in [0,1]$ and are asked to partition $\{1, 2, \ldots, n\}$ into $S_1, S_2, \ldots, S_k$ such that $\sum_{i \in S_j} x_i \leq 1$ for $j = 1, 2, \ldots, k$ and such that $k$ is as small as possible. The elements $i \in S_j$ are thought of as being placed in bin $j$ which has capacity 1. Then $k$ is the number of bins used.

The analysis of bin packing algorithms has proved to be very challenging. There are many deep results and the reader is referred to a survey by Coffman and Johnson [27] for further reading.

We now give an accessible result due to Frederickson [39]. Suppose that $x_1, x_2, \ldots, x_n$ are independent uniform [0,1] random variables. It is clear that the number of bins required is at least $\sum_{j=1}^{n} x_j$ which by Hoeffding's inequality is **whp** in the range $\frac{n}{2} + o(n)$. We describe an algorithm FOLD for which the expected number of bins used is at most $\frac{n}{2} + 2n^{2/3}$

Let $\alpha = 1 - n^{-1/3}$.

1. Place each element $x_i \geq \alpha$ into a bin on its own. Suppose there are $B_1$ such.
2. Let $N = n - B_1$ be the number of bins remaining to be packed.
3. Order the items so that $x_1 \leq x_2 \leq \cdots x_N \leq \alpha$.
4. For $i = 1, 2, \ldots, \lfloor N/2 \rfloor$
   **(a)** Put $x_i, x_{N-i+1}$ into one bin if $x_i + x_{N-i+1} \leq 1$.
   **(b)** Put $x_i, x_{N-i+1}$ into separate bins if $x_i + x_{N-i+1} > 1$.

   Put item $\lceil N/2 \rceil$ into a separate bin if $N$ is odd.

**Theorem 7** *The expected number of bins packed by FOLD is at most* $\frac{n}{2} + 2n^{2/3}$.

Let $B_2$ be the number of bins used in Step 4(b). Now $B_1$ is distributed as the binomial $B(n, 1-\alpha)$ and so

$$\mathbf{E}(B_1) = n^{2/3}.$$

Let $W_i = x_i + x_{N-i+1}$ for $i = 1, 2, \ldots, \lfloor N/2 \rfloor$. A calculation gives

$$\mathbf{Pr}(W_i > 1) \leq \frac{2i}{(N+1)(N+2)(1-\alpha)^2} \leq \frac{1}{N(1-\alpha)^2} = \frac{n^{2/3}}{N}.$$

Thus $\mathbf{E}(B_2) \leq n^{2/3}$. Now the number of bins $B$ used by FOLD satisfies

$$B = \frac{n}{2} + B_1 + \frac{B_2}{2} + (N - 2\lfloor N/2 \rfloor)$$

and the theorem follows. □.

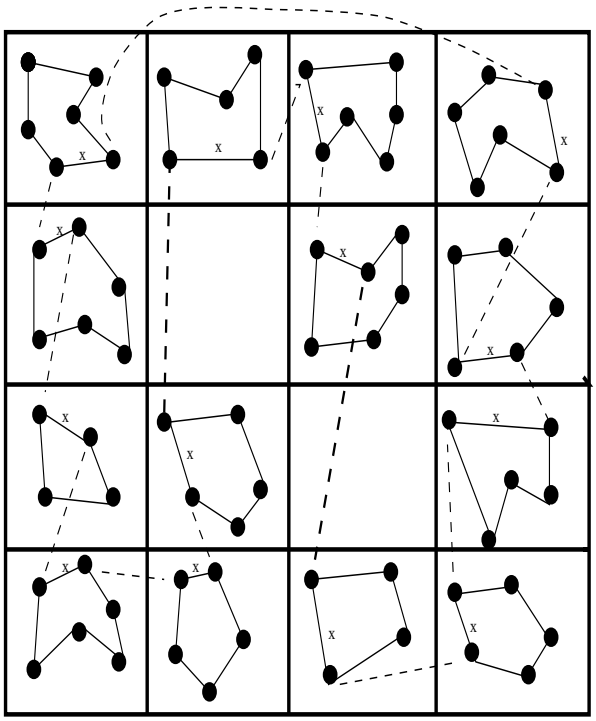
## 4.2 Euclidean Travelling Salesman Problem

One of the earliest and most influential results in the probabilistic analysis of combinatorial optimization problems was Karp's partitioning algorithm [61] for the travelling salesman problem in the unit square $C = [0,1]^2$. Here we have $n$ points $X_1, X_2, \ldots, X_n$ chosen uniformly at random in $C$ and the problem is to find the minimum length tour (i.e. Hamilton cycle) through them, using Euclidean distance to define the distance between points.

Let $\ell^* = \ell^*(X_1, X_2, \ldots, X_n)$ be the minimum length of a tour. We give an outline of a simplified version of Karp's algorithm. First we mention the equally important results of Beardwood, Halton and Hammersley [7]. Their results are stronger and more general, but in any case they imply that there exists an (unknown) constant $\beta > 0$ such that for any $\epsilon > 0$

$$\lim_{n\to\infty} \mathbf{Pr}\left( \left| \frac{\ell^*}{\sqrt{n}} - \beta \right| > \epsilon \right) = 0.$$

In other words we expect that $\ell^* \approx \beta\sqrt{n}$. Consider the following heuristic:



Patch by adding broken edges and deleting edges marked with an x

Figure 1: Patching Sub-Tours

**Partitioning Algorithm**

**(a)** Divide $C$ into $M = m^2$ squares $C_1, C_2, \ldots, C_M$ of size $\frac{1}{m} \times \frac{1}{m}$ where $m = \epsilon\sqrt{n}$ for some small $\epsilon > 0$..

**(b)** Find an optimal tour $T_i$ through the points $A_i$ in each $C_i$.

**(c)** Patch these tours together to make a tour $\hat{T}$ as indicated in Figure 1.

Let $T^*$ be the optimum tour and let $\ell_i^*$ be the length of the edges and parts of edges of $T^*$ which lie in $C_i$. One can patch these edges to a tour of $A_i$, see Figure 2, at an additional cost of at most the perimeter of $C_i$. Therefore

$$\ell_i^* \geq \ell(T_i) - \frac{4}{m} \qquad\qquad 1 \leq i \leq M. \tag{4}$$

The length of the tour $\hat{T}$ obtained by the patching satisfies

$$\ell(\hat{T}) \leq \sum_{i=1}^{M} \ell(T_i) + 6m. \tag{5}$$

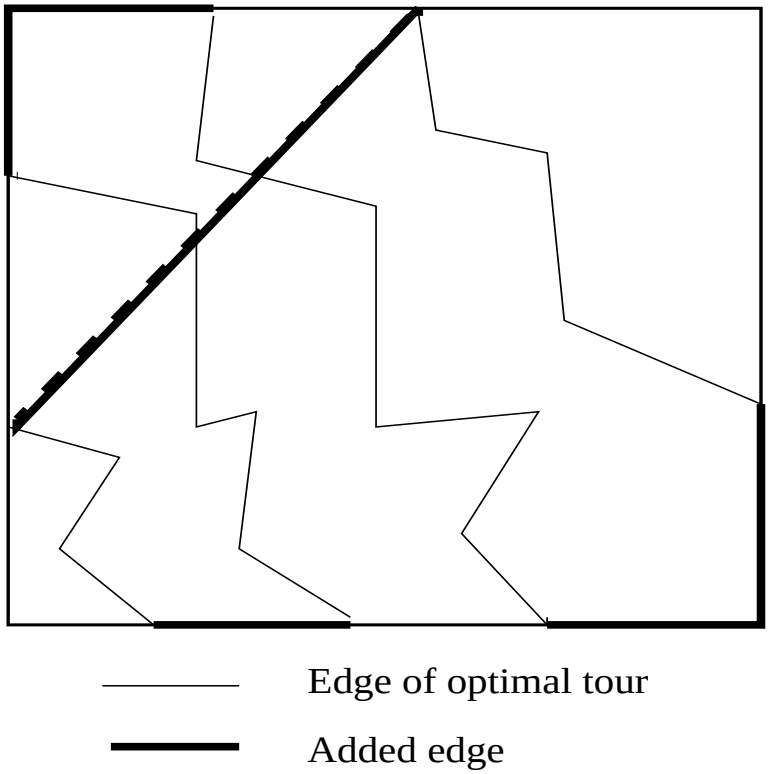



Figure 2: Connecting Pieces of Optimal Tour in Sub-Square

It follows from (4) and (5) that

$$\ell^* \leq \ell(\hat{T}) \leq \ell^* + 10\epsilon\sqrt{n}.$$

Since $\ell^* \approx \beta\sqrt{n}$ **whp** we see that $\hat{T}$ is asymptotically optimal.

How long does it take to compute $\hat{T}$? Each tour $T_i$ can be computed in time $O(|A_i|^2 2^{|A_i|})$ by dynamic programming. Now $|A_i|$ has distribution $B = Bin(n, 1/M)$ and so the expected running

time for computing all the $T_i$'s is of order

$$\begin{aligned}
\mathbf{E}\left(\sum_{i=1}^{M}|A_i|^2 2^{|A_i|}\right) &= M\mathbf{E}(B^2 2^B) \\
&= M\sum_{k=0}^{n}\binom{n}{k}k^2 2^k M^{-k}\left(1-\frac{1}{M}\right)^{n-k} \\
&\leq 2M\left(1-\frac{1}{M}\right)^{n}\sum_{k=2}^{n}\binom{n}{k}k(k-1)\left(\frac{2}{M-1}\right)^{k}+2e^{-\epsilon^{-2}}n \\
&\leq \frac{2}{n}M\sum_{k=2}^{n}n(n-1)\binom{n-2}{k-2}\left(\frac{2}{M-1}\right)^{k}+2e^{-\epsilon^{-2}}n \\
&= \frac{2(n-1)}{(M-1)^2}\left(1+\frac{2}{M-1}\right)^{n-2}+2e^{-\epsilon^{-2}}n \\
&\leq 3\epsilon^{-2}e^{\epsilon^{-2}}n.
\end{aligned}$$

This constitutes the main amount of work and so in expected time $O(\epsilon^{-2}e^{\epsilon^{-2}}n)$ we can find a solution which is likely to be within $1+O(\epsilon)$ of optimal.

Since the appearance of [61] and [7] there has been a great amount of research effort devoted the analysis of optimization problems in Euclidean space. A recent book by Steele [85] is an excellent source for this material.

## 4.3 Asymmetric Travelling Salesman Problem

The *Assignment Problem (AP)* is the problem of finding a minimum-weight perfect matching in an edge-weighted bipartite graph. An instance of the AP can be specified by an $n\times n$ matrix $M=(m_{ij})$; here $m_{ij}$ represents the weight of the edge between $x_i$ and $y_j$, where $X=\{x_1,x_2,\ldots,x_n\}$ is the set of "left vertices" in the bipartite graph, and $Y=\{y_1,y_2,\ldots,y_n\}$ is the set of "right vertices."The AP can be stated in terms of the matrix $M$ as follows: find a permutation $\sigma^*=\sigma^*(M)$ of $\{1,2,\ldots,n\}$ that minimizes $\sum_{i=1}^{n}m_{i,\sigma(i)}$. Let $AP(M)$ be the optimal value of the instance of the AP specified by $M$.

The *Asymmetric Traveling-Salesman Problem (ATSP)* is the problem of finding a Hamiltonian circuit of minimum weight in an edge-weighted directed graph. An instance of the ATSP can be specified by an $n\times n$ matrix $M=(m_{ij})$ in which $m_{ij}$ denotes the weight of edge $<i,j>$. The ATSP can be stated in terms of the matrix $M$ as follows: find a cyclic permutation $\pi^*=\pi^*(M)$ of $\{1,2,\ldots,n\}$ that minimizes $\sum_{i=1}^{n}m_{i,\pi(i)}$; here a cyclic permutation is one whose cycle structure consists of a single cycle. Let $ATSP(M)$ be the optimal value of the instance of the ATSP specified by $M$.

It is evident from the parallelism between the above two definitions that $AP(M)\leq ATSP(M)$. The ATSP is NP-hard, whereas the AP is solvable in time $O(n^3)$.

Karp [62] studied the relationship between $AP$ and $ATSP$ when entries of the matrix $M$ are independent [0,1] uniform random variables. He proved the rather surprising result that

$$\mathbf{E}(ATSP(M)) \leq \mathbf{E}(AP(M)) + o(1).$$

The proof was quite involved and later on Karp and Steele [64] simplified the argument and improved the error term. Subsequently, Dyer and Frieze [34] reduced the error term to $O((\log n)^4/\log\log n)$. We give an outline of the approach from [64].

The first important observation is that the solution $\sigma^*$ of $AP(M)$ will be a random permutation.

$$\mathbf{Pr}(\sigma^*(M)) = \sigma_1) = \mathbf{Pr}(\sigma^*(\sigma_2 M)) = \sigma_2\sigma_1) = \mathbf{Pr}(\sigma^*(M)) = \sigma_2\sigma_1)$$

where $\sigma M$ is the matrix obtained by permuting the columns of $M$ by $\sigma$. Note that $M$ and $\sigma M$ have the same distribution. Thus **whp** the optimal solution $\sigma^*$ will have $O(\log n)$ cycles. See e.g. Bollobás [12].

Karp and Steele then argue that **whp** the optimal solution to $AP(M)$ does not contain any edges of length greater than $\lambda = K(\log n)^2/n$ for some suitably large constant $K > 0$. Thus if we remove the edges of length greater than $\lambda$ from the problem before solving $AP(M)$ then **whp** we will get the same solution. This means that we can pessimistically consider the edges not in the optimal assignment solution to independently have length uniform in $[\lambda, 1]$ as we defer specifying their exact length until after solving the AP.

Suppose that the solution to $AP(M)$ consists of cycles $C_1, C_2, \ldots, C_k$ where $|C_1| \geq |C_2| \geq \cdots \geq |C_k|$ where $|C_1| = \Omega(n/\log n)$. The idea is to iteratively *patch* $C_{i+1}$ into a cycle $\hat{C}_i$ formed on the vertices of $C_1 \cup C_2 \cup \cdots \cup C_i$.

A patch involves deleting an edge $xy$ of $C_{i+1}$ and an edge $uv$ of $\hat{C}_i$ and replacing them by the edges $xv, uy$ to create a single cycle. The algorithm chooses the patch which minimises the cost $m_{xv} + m_{uy}$. If $|\hat{C}_i| = a$ and $|C_{i+1}| = b$ and $Z_i$ denotes the cost of the best patch, then for any $\xi > 0$

$$\mathbf{Pr}(Z_i > 2\xi + 2\lambda) \leq (1-\xi^2)^{ab}.$$

This is because if $Z_i \geq 2\xi + 2\lambda$ then for every relevant $x, y, u, v$ it is *not* the case that $m_{xv} \leq \xi + \lambda$ and $m_{uy} \leq \xi + \lambda$. In our pessimistic model these events can be considered independent as they deal with disjoint sets of edges. Now by assumption $ab = \Omega(n/\log n)$ and so

$$\mathbf{Pr}(\exists i :\ Z_i \geq (\log n)/n^{1/2}) = o(1).$$

**Whp** there are $O(\log n)$ cycles and so **whp** the total patching cost is $O((\log n)^2/n^{1/2})$.

## 4.4 Disjoint paths

Suppose we are given a graph $G = (V, E)$ and a set of pairs $(a_i, b_i)$, $1 \leq i \leq K$ of vertices. In the Edge Disjoint Paths Problem (EDPP) we want to find paths $P_i$ joining *source* $a_i$ to *sink* $b_i$

for $1 \le i \le K$ which are edge disjoint, or prove it is not possible. In the Vertex Disjoint Paths Problem (VDPP), the vertices are all distinct and we want vertex disjoint paths. Both problems are solvable in polynomial time if $K$ is fixed, independent of the input, Robertson and Seymour [79], but NP-hard if $K$ varies. The problem is interesting for theoretical and practical reasons; the latter interest comes from its use as a model for some communications problems.

For random graphs $G_{n,m}$ the VDPP was considered by Shamir and Upfal [81] who gave a linear time algorithm which **whp** succeeds in finding paths provided $m \ge 2n \log n$ and $K = O(\sqrt{n})$. It should be remarked that here the two sets of vertices are fixed *before* the random graph is constructed. The problem was also considered by Hochbaum [56] who gave a $o(m)$ time algorithm when $K = O(\sqrt{d/\log n})$, $d = 2m/n$ is the average degree. Both algorithms are based on growing disjoint forests rooted at the sources and sinks until the corresponding trees are large enough so that for each $i$ the tree rooted at $a_i$ can be joined to the tree rooted at $b_i$.

The above approach is simple and efficient, but does not address the problem when the random graph is constructed first and then the sources and sinks are chosen by an *adversary*. Suppose $2m/n - \log n \to \infty$ so that $G_{n,m}$ is connected **whp**. Let $D$ be the median distance between pairs of vertices in $G_{n,m}$. Then $D = O(\log n/\log d)$ **whp**. Clearly it is not possible to connect more than $O(m/D)$ pairs of vertices by edge-disjoint paths, for all choices of pairs, since some choice would require more edges than all the edges available. Also, some restriction on the number of times a vertex can be a source or sink is necessary. Thus the following theorem of Broder, Frieze, Suen and Upfal [19] is optimal up to constant factors.

**Theorem 8** *Suppose $2m/n - \log n \to \infty$. Then there exist positive constants $\alpha$ and $\beta$ such that* **whp**, *for all $A = \{a_1, a_2, \dots, a_K\}, B = \{b_1, b_2, \dots, b_K\} \subseteq [n]$ satisfying*

*(i)* $K = \lceil \alpha m \log d/\log n \rceil$,

*(ii) for each vertex $v$,* $|\{i : a_i = v\}| + |\{i : b_i = v\}| \le \min\{d_G(v), \beta d\}$,

*there exist edge-disjoint paths in $G_{n,m}$, joining $a_i$ to $b_i$, for each $i = 1, 2, \dots, K$. Furthermore, there is an $O(nm^2)$ time randomized algorithm for constructing these paths.*

The strategy for proving Theorem 8 is quite different from [81] and [56]. First of all the sources and sinks are joined, by a network flow algorithm, to randomly chosen $\tilde{a}_i, \tilde{b}_i$, $1 \le i \le K$. This has a *spreading out* effect, similar to that achieved by the method of Valiant and Brebner [89] for routing messages in the $n$-cube. The new sources and sinks are then joined up by utilizing random walks.

Frieze and Zhao [49] have extended the above ideas to deal with random $r$-regular graphs where $r$ is considered to be constant.

The VDPP is discussed in [20]. Using similar ideas to those above it is shown that:

**Theorem 9** *Suppose* $2m/n - \log n \to \infty$. *Then there exist positive constants* $\alpha, \beta$ *such that* **whp**, *for all* $A = \{a_1, a_2, \ldots, a_K\}, B = \{b_1, b_2, \ldots, b_K\} \subseteq [n]$ *satisfying*

**(i)** $A \cap B = \emptyset$,

**(ii)** $|A| = |B| = K \leq \frac{\alpha n \log d}{\log n}$,

**(iii)** $|N(v) \cap (A \cup B)| \leq \beta |N(v)|, \quad \forall v \in V$,

*there are vertex disjoint paths* $P_i$ *from* $a_i$ *to* $b_i$ *for* $1 \leq i \leq K$. *Furthermore, there is an* $O(nm^2)$ *time randomized algorithm for constructing these paths.*

Here $N(v)$ is the neighbour set of vertex $v$. This is again optimal up to the constant factors $\alpha, \beta$.

# 5 Greedy Algorithms

In this chapter, we continue to focus on the average performance guarantees of algorithms which are sure to run in polynomial time. In particular, we focus on the expected behaviour of greedy algorithms. These algorithms are appealing because they are usually fast and easy to implement. we consider three examples, a greedy algorithm for constructing a stable set, a greedy algorithm for constructing a matching, and a greedy algorithm for the Knapsack problem.

## 5.1 Cliques, Stable Sets, and Colourings

We consider the following greedy algorithm for constructing a stable set. Pick a vertex $x$, determine which vertices are not adjacent to $x$, recursively apply the algorithm to find a stable set $S$ in the graph induced by these vertices, and return $S + x$.

We prove:

**16 Whp** *the above algorithm finds a stable set of size at least* $\log_2 n - 3log_2 log_2 n$ *in* $G_{n,\frac{1}{2}}$.

**Proof** The algorithm terminates with a stable set $S$ such that every vertex of $G - S$ sees a vertex of $S$. But it is easy to compute that the number of such sets (stable or otherwise) with fewer than the given number of vertices is $o(1)$. □

For a sharper analysis, see []. Now, a classic result due to ** states that

**17 Whp** *the largest stable set in* $G_{n,\frac{1}{2}}$ *has* $2\log_2 n - * * - O(1)$ *elements.*

Thus the algorithm typically constructs a stable set which is about half the size of the largest stable set.

We can analyze our algorithm using the method of deferred decisions. We note that in constructing the stable set we need only examine edges which have an endpoint in the stable set. It follows that $G_{n,\frac{1}{2}} - S$ is a uniformly chosen random graph on vertex set $V_n - S$. So, we can re-apply our algorithm to rip out a stable set disjoint from $S$. Repeating this procedure allows us to colour $G$ with $(1+o(1))\frac{n}{logn}$ colours. A beautiful analysis due to Bollobas shows:

**18 Whp** *the chromatic number of* $G_{n,\frac{1}{2}}$ *is* $(1+o(1))\frac{n}{2\log_2 n}$.

Thus our colouring algorithm uses about twice the optimal number of colours. To close this section, we mention two open problems.

**Research Problem** Develop a polynomial-time algorithm which finds a stable set of size $(\frac{1}{2}+\epsilon)log_2 n$ in $G_{n,\frac{1}{2}}$ **whp**, for some constant $\epsilon > 0$.

**Research Problem** Develop a polynomial-time algorithm which finds a colouring of $G_{n,\frac{1}{2}}$ using $(1-\epsilon)\frac{n}{log_2 n}$ colours **whp**, for some constant $\epsilon > 0$.

## 5.2 Greedy Matchings

In this section we consider finding perfect matchings in *sparse random graphs*. Recall that the random graph $G_{n,m}$ has vertex set $\{1,2,\ldots,n\}$ and $m$ random edges. The graph is considered to be *sparse* if $m = \lceil cn \rceil$ for some constant $c > 0$. In this case $G_{n,m}$ has no perfect matching **whp**. We leave it as an exercise to show that **whp** there are a large number of isolated vertices. This is an interesting case, because as we have seen, it is easy to find a perfect matching when there are many more edges. For such a *sparse* random graph the interest is in using a simple heuristic to find a large matching which is close to optimal **whp**. Researchers have concentrated in the main on the analysis of greedy heuristics:

**GREEDY**

**begin**
  $M \leftarrow \emptyset$;
  **while** $E(G) \neq \emptyset$ **do**
  **begin**
    ***A***: Choose $e = \{u,v\} \in E$
      $G \leftarrow G \setminus \{u,v\}$;
      $M \leftarrow M \cup \{e\}$
  **end**;
Output $M$
**end**

($G \setminus \{u,v\}$ is the graph obtained from $G$ by deleting the vertices $u, v$ and all edges incident with them, together with any vertices which become isolated.)

The average performance of GREEDY when the input is random was first analysed by Tinhofer [88]. He considered its performance on the random graph $G_{n,p}$ in the dense case where $p$ is fixed independent of $n$. In this case it is fairly easy to show that the algorithm produces a matching of size $n/2 - O(\log n)$ **whp**. In fact the analysis in Section 3.1 essentially yields this result.

Let $X = X(n,m)$ be the random number of edges in the matching produced by GREEDY applied to $G_{n,m}$ when the edge choice in statement **A** is uniformly random. Dyer, Frieze and Pittel [37] were able to establish the asymptotic distribution of this variable when $m = \lfloor cn \rfloor$. In particular they showed that $\mathbf{E}(X) \approx \phi(c)n$, where $\phi(c) = \frac{c}{2(c+1)}$ (and that this variable is asymptotically normal).

It is possible to modify this algorithm without considerable complications, so as to improve its likely performance. Perhaps the simplest modification is to first choose a vertex $v$ at random and then to randomly choose an edge incident with $v$. We refer to this as MODIFIED GREEDY. Dyer, Frieze and Pittel also analysed the performance of MODIFIED GREEDY in the same setting as for GREEDY. Let $\hat{X} = \hat{X}(n,m)$ be the random number of edges in the matching produced by MODIFIED GREEDY on $G_{n,m}$. Now the asymptotic expectation increases to $\mathbf{E}(\hat{X}) \approx \hat{\phi}(c)$ where $\hat{\phi}(c) = \frac{1}{2} - \frac{\log(2-e^{-c})}{2c} > \phi(c)$.

GREEDY and MODIFIED-GREEDY both find matchings which are less than the maximum by a constant factor. Karp and Sipser [63] considered a similar greedy type of algorithm which we will call KSGREEDY. Their algorithm (a) chooses an edge incident to a vertex of degree 1 while there is one and otherwise (b) chooses a random edge. The algorithmic change is tiny, but the improvement in performance is spectacular. They show that this algorithm is asymptotically optimal in the sense that with high probability it finds a matching which is within $o(n)$ of the optimum size! They also prove that if $c \leq e$ then KSGREEDY spends almost all of its time in case (a). The algorithm is considered to run in two phases. Phase 1 ends when the minimum degree of the graph that remains is at least two. Note that during Phase 1 the algorithm makes correct choices in the sense that the edges chosen are a subset of some maximum matching.

Aronson, Frieze and Pittel [6] have undertaken a further analysis of this algorithm.

- If $c < e$ then at the end of Phase 1, all that is left of the graph is a few vertex disjoint cycles.
- If $c > e$ then in Phase 2, KSGREEDY will match all but about $n^{1/5}$ of those vertices which remain at the end of Phase 1. More precisely, there exist positive constants $c_1, c_2, a, b$ such that if $L$ denotes the number of vertices which become isolated in Phase 2, then

$$c_1 n^{1/5} (\log n)^{-a} \leq \mathbf{E}(L) \leq c_2 n^{1/5} (\log n)^{b}. \tag{6}$$

- Analysis of the algorithm gives an asymptotic expression for the size of the maximum matching in $G_{n,m}$.

Another possible version of GREEDY is MINGREEDY where in Step A one chooses a (random) vertex of minimum degree and then a random neighbour of this vertex. Frieze, Radcliffe and Suen [46] considered the performance of MINGREEDY on random cubic graphs (a graph is cubic if every vertex has degree three). They proved

**Theorem 10** *Let $L_n$ denote the number of vertices left exposed by the matching constructed by running MINGREEDY on a random cubic graph with $n$ vertices. Then there exist constants $d_1, d_2 > 0$ such that*

$$d_1 n^{1/5} \leq \mathbf{E}(L_n) \leq d_2 n^{1/5} \log n. \tag{7}$$

We note that a random cubic graph has a perfect matching **whp**, see for example Bollobás [12].

Thus MINGREEDY usually does very well. Note the common exponent 1/5 in (6) and (7). This can be explained to some extent by the fact that near the end of KSGREEDY, when most *avoidable* vertex isolations are made, the maximum degree is bounded **whp**.

In computational experiments MINGREEDY left an average of just over 10 vertices unmatched when run on random cubic graphs with $10^6$ vertices.

## 5.3 Knapsack Problems

In this section we consider the 0-1 Knapsack problem in which we have $n$ items $I_1, ..., I_n$, some subset of which we shall put in a knapsack. Each item $I_i$ has an associated weight $w_i$ and profit $p_i$. Our restriction is that the knapsack can hold total weight at most $W$ and our objective is to maximize the profit. That is, we solve:

$$\text{Maximise} \quad \sum_{j=1}^{n} p_j x_j \tag{8}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} w_j x_j \leq W \tag{9}$$

$$x_j = 0/1 \qquad 1 \leq j \leq n$$

Here we analyze a random instance in which the coefficients $p_1, \ldots, p_n, w_1, \ldots, w_n$ are independently chosen from the unit interval [0,1]. For the constraint (9) to be active but not too strong we let $W = \beta n$ where $0 < \beta < 1/2$. The following greedy algorithm is likely to have a good asymptotic average performance..

**Greedy**
**begin**
Order the variables in increasing order of value $p_j/w_j$.
$S := 0; x_j := 0$ **for** $j = 1$ **to** $n$;

**For** $j = 1$ **to** $n$ **do**
**begin**
**If** $w_j \leq W - S$ then $x_j := 1; S := S + w_j$
**end**
**end**

The algorithm is known to produce at least a 1/2-optimal solution, but is likely to do much better. Let $Z^*$ denote the optimal value in (8), $Z_{LP}$ the optimal solution to the Linear Programming relaxation and $Z_G$ the value of the solution produced by Greedy. Then

$$Z^* \geq Z_G \geq Z_{LP} - 1 \geq Z^* - 1. \tag{10}$$

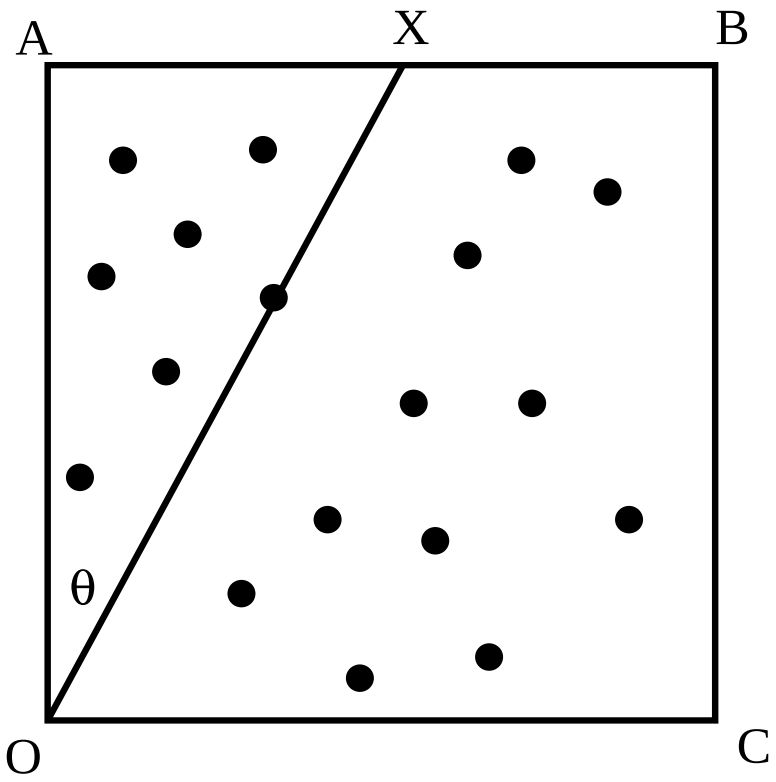
Now, assuming $w_1 + w_2 + \cdots + w_n > W$ (and this is true **whp**)

$$Z_{LP} = \sum_{j=1}^{t} p_j + \alpha p_{t+1}$$

where $0 \leq \alpha < 1$ and

$$\sum_{j=1}^{t} w_j + \alpha w_{t+1} = W < \sum_{j=1}^{t+1} w_j.$$

There is a geometric interpretation:



The pairs $(w_j, p_j)$ are chosen uniformly from the unit square OABC. We sweep the line OX clockwise starting at OA until we have swept over points whose $w$ sum exceeds $W$. Then we stop with $OX$ through a point $(w_f, p_f)$ where $x_f = \alpha$.

Now consider a fixed $\theta$ and let $A_\theta$ denote the area of the region $T_\theta$ to the left of $OX$.

$$A_\theta = \begin{cases} \frac{\tan\theta}{2} & 0 \le \theta \le \pi/4 \\ 1 - \frac{\coth\theta}{2} & \pi/4 \le \theta \le \pi/2 \end{cases}$$

Next let $w_\theta$ denote the expected $w$ coordinate of a point chosen uniformly at random within $T_\theta$ and let $p_\theta$ be the corresponding expected $p$ coordinate.

$$w_\theta = \begin{cases} \frac{\tan\theta}{3} & 0 \le \theta \le \pi/4 \\ \frac{C^2-3C+3}{3(2-C)} & \pi/4 \le \theta \le \pi/2 \qquad C = \coth\theta \end{cases}$$

and

$$p_\theta = \begin{cases} \frac{2}{3} & 0 \le \theta \le \pi/4 \\ \frac{2C^3-3C^2+3}{3(2-C)} & \pi/4 \le \theta \le \pi/2 \end{cases}$$

The expected weight $w(T_\theta)$ of points falling in $T_\theta$ is $nA_\theta w_\theta$. Define $\theta_0$ by $A_{\theta_0} w_{\theta_0} = \beta$. Applying Hoeffding's inequality we see that for any $\theta$

$$\mathbf{Pr}(|w(T_\theta) - nA_\theta w_\theta| \ge t) \le 2e^{-2t^2/n}$$

and

$$\mathbf{Pr}(|p(T_\theta) - nA_\theta p_\theta| \ge t) \le 2e^{-2t^2/n}$$

It follows that **whp**

$$Z_{LP} = nA_{\theta_0} p_{\theta_0} + O(\omega n^{1/2}) \tag{11}$$

for any $\omega \to \infty$.

It follows from (10) and (11) that **whp** $Z_G$ is a good approximation to $Z^*$.

This is fairly simple. Lueker [72] proved a much deeper result.

$$\mathbf{E}(Z_{LP} - Z^*) = O((\log n)^2/n).$$

He did this basically by showing that **whp** there exists a good integer solution obtainable by changing a few ($O(\log n)$) values of $x_j$ in the optimal linear program. Goldberg and Marchetti-Spaccamela [52] used this to define a simple enumerative search with the following property: for any $\epsilon > 0$ there is an $O(n^{d(\epsilon)})$ time algorithm which solves this model of a knapsack problem *exactly* with probability at least $1 - \epsilon$.

Subsequently Dyer and Frieze [33, 35] extended this approach to multi-dimensional knapsack problems and generalised assignment problems with a bounded number of constraints.

Mamer and Schilling [73] established probabilistic approximation results for multi-dimensional knapsack problems with the number of constraints growing with $n$.

**Related problems**
In the **Subset-Sum** problem we are given $a_1, a_2, \ldots, a_n, b$ and asked to decide if there exists a subset $S \subseteq \{1, 2, \ldots, n\}$ such that $a(S) = \sum_{i \in S} a_i = b$. This has some cryptographic applications, [75]. Lagarias and Odlyzko [69] gave a lattice based algorithm for solving this problem when the $a_i$ are chosen independently from $\{1, 2, \ldots, 2^{n^2}\}$ and $b = \sum_{i \in S^*} a_i$ for some unknown set $S^*$. Frieze [42] gave a simplified analysis of their result.

In the **Partition** problem we are given $a_1, a_2, \ldots, a_n$ and asked to find the set $S$ which minimises $|a(S) - a(\bar{S})|$. Assume that $a_1, a_2, \ldots, a_n$ are chosen independently and uniformly from [0,1]. It is known that **whp** this minimum is of order $n2^{-n}$, see Karmarker, Karp, Lueker and Odlyzko [60]. On the other hand, Karmarker and Karp [59] gave an algorithm which **whp** finds a set $S$ with $|a(S) - a(\bar{S})| \leq (\log n)^{-c \log n}$ for some constant $c > 0$. They gave another more elegant and natural algorithm and conjectured that it had the same performance. This was recently verified by Yakir [91].

# 6 Negative Results

In this chapter, we focus on results which show that algorithms are typically inefficient or that problems are usually hard. Actually, we devote almost all of our discussion to the first of these topics. To begin we present a proof that a certain branch and bound algorithm for the knapsack problem takes super-polynomial time **whp** on a random example drawn from a specific probability distribution. we then present less detailed discussions of similar results for the quadratic assignment problem and the $k$-median problem. Finally, we survey some other results in this vein.

Showing that problems are difficult on average is much more difficult than showing that a certain algorithm is typically inefficient. In particular, if we show that an NP-complete problem is difficult on average then we can deduce that $P \neq NP$. The best we can hope for is to prove "on-average" completeness results analogous to those developed for NP. This theory is outside the scope of this paper, and uses a very different notion of "average". For these reasons, we content ourselves with giving the address of a web-site dedicated to the theory, and a quote from some introductory material posted on the web-site. The web-site is:

http://www.uncg.edu/mat/avg.html

The quote is:

> Despite many years of intensive effort, there are no known efficient algorithms for NP-complete problems, where by efficient we mean algorithms that are fast in the worst case. Due to this striking gap in our knowledge, the search for algorithms

that are "efficient" according to various more modest criteria has attracted increasing attention.

One particularly interesting criterion is that of requiring problems be solvable quickly "on average." That is, can one solve NP-complete problems via algorithms that, although possibly very slow on some inputs, are fast on average with respect to some underlying probability distributions on instances. Algorithms that are fast on average have been found for several NP-complete problems, such as the vertex -coloring problem and the Hamiltonian path problem, under commonly used distributions on graphs.

However, there also are NP-complete problems that have so far resisted such "average case" attacks. Are these problems difficult on average? What does it mean for a problem to be difficult on average, and how is one to know whether a problem is difficult on average? In his seminal paper[**?**] , Levin initiated the study of these questions. Two fundamental and robust notions were defined along lines similar to (standard, worst-case) NP-completeness theory. Namely, he introduced the notion of average polynomial time for measuring "easiness" on average and the notion of average-case NP-completeness for measuring "hardness" on average. Levin then showed that a tiling problem is average-case NP-complete if each parameter of an instance is randomly selected. This framework has been studied and enhanced by a number of researchers and several more average-case NP-complete problems have been found. Such average-case completeness results, as indicated by Levin [**?**], may not only save misguided "positive" efforts-such as trying to find fast-on-average algorithms for problems that probably lack them-but might also be used in areas (like cryptography) where hardness on average of some problems is a frequent assumption.

## 6.1 Knapsack

The simplest method for solving a 0-1 Knapsack problem is to compute the weight and cost of each subset of the items and choose the highest profit subset that fits in the knapsack. We can enumerate all these possible solutions in a systematic way with the aid of a complete binary tree of height $n$ as shown in Fig. **. Each path of the tree from the node to the route corresponds to a partial solution where if we branch right at height $i$ then item $i$ is in the solution and if we branch left at height $i$ it is not.

More generally, we can construct an enumeration tree $T$ which is a complete binary tree of height $n$ such that

(i) every node $s$ corresponds to a partial solution consisting of a subset $S_s$ of the items and a partition of $S_s$ into two sets $P_s$, those which we intend to put into the knapsack, and $Q_s$, those which we do not intend to put in the knapsack.

(ii) If $r$ is the root of the tree $S_r$ is empty, and for each non-leaf node $s$ with right child $s^r$ and left child $s^l$ there is an item $I_s$ not in $S_s$ such that $S_{s^r} = S_{s^l} = S_s + I_s$, $P_{s^l} = P_s$, $P_{s^r} = P_s + I_s$.

See Fig. ** for an example: Thus, in our original enumeration tree we insisted that if two nodes $s$ and $t$ have the same level then $I_s = I_t$, a condition we now drop without losing the bijection between the leaves and the subsets of the items.

Now, in generating all the candidate solutions, we do not need to construct the whole tree. For example, if there is a node $s$ such that $\Sigma_{I \in P_s} w(I) > B$ then for every leaf $l$ in the subtree $T_s$ underneath $s$, since $P_s \subseteq P_l$, $P_l$ does not fit in the knapsack so there is no point exploring $T_s$. More generally, there is no point in exploring the subtree underneath a node if we know there is no optimal solution underneath this node.

In a branch and bound algorithm for the 0-1 knapsack problem, we generate some partial subtree of a complete enumeration tree whilst ensuring that one of its leaves corresponds to an optimal solution. We begin with the root, and repeatedly *branch* out from the tree constructed so far by adding two children at some leaf $l$. Throughout the algorithm, we have a set of active leaves of the current tree, which are those underneath which we intend to search. We must ensure that at all times, there is some optimal solution lying in a subtree underneath an active leaf. Initially, the root is active, and when we branch (from an active leaf), the two new leaves become active. We may make a leaf $l$ inactive for either of the following two reasons.

(i) An already explicitly computed solution has at least as good a solution value as the best solution in $T_l$, or

(ii) There is another active leaf $l'$ such that for any solution corresponding to a leaf of $T_l$ there is a leaf of $T_{l'}$ which corresponds to a solution which is at least as good.

We continue growing the partial enumeration tree, as long as there are any active leaves which are not also leaves of the complete enumeration tree, making leaves inactive whenever we can. Obviously, the best solution corresponding to a leaf of our partial tree is an optimal solution to the knapsack problem. Our hope is that the pruning due to (i), (ii), and a clever choice of the items on which we choose to branch, will restrict the partial tree to a reasonable size.

We remark that this technique clearly generalizes to other optimization problems. In particular, it is often applied to 0-1 programming problems, in which case to compute a bound on the best possible solution in $T_l$ we usually consider the fractional relaxation of the integer program. For example, we remark that in our knapsack problems, for any node $s$ of the partial tree, a solution corresponding to a leaf of $T_s$ has profit at most $B_s = \sum_{I_i \in P_s} p_i + (B - \sum_{I_i \in P_s} w_i) max_{I_i \notin P_s} (\frac{p_i}{w_i})$ , because any fractional solution with $x_i = 1$ for each $I_i \in P_s$ will generate at most this much profit. Thus, if $B_s$ is less than the profit of the optimal solution found so far, then we can make $s$ inactive. The results in Section 5.3 can be reinterpreted as stating that using this pruning

procedure, and always branching so as to maximize $\frac{p_i}{w_i}$ for the item $I_i$ on which we branch, for sufficiently small $\epsilon$, we obtain the optimal solution in polynomial time with probability $1 - \epsilon$.

We turn now to a specific 0-1 knapsack problem and a refinement of this branch and bound algorithm. We insist that the weights and costs and $B$ are all integers. We note that in this case, we can improve the above remark and obtain:

**19** *For any node $s$ of the partial tree, let $d$ be the greatest common divisor of the weights of the items not in $P_s$. Then a solution corresponding to a leaf of $T_s$, has profit at most $C_s = \sum_{I_i \in P_s} p_i + d \left\lfloor \frac{(B - \sum_{I_i \in P_s} w_i)}{d} \right\rfloor \max_{I_i \notin P_s} (\frac{p_i}{w_i})$.*

We denote by $OPT$ the best solution found to date by the algorithm. We will make a node $l$ inactive if:

(A) $\Sigma_{I_i \in P_l} w_i > B$,or

(B) $C_l \leq Opt$, or

(C) there is an active leaf $l'$ such that $S_l = S_{l'}$, or $\Sigma_{I_j \in P_l} w_j \geq \Sigma_{I_j \in P_{l'}} w_j$, and $\Sigma_{I_j \in P_l} p_j \leq \Sigma_{I_j \in P_{l'}} w_j$

We remark that for any $l, l'$ as in $B$, if $P_l + X$ is the set of items put in the knapsack for some feasible solution corresponding to a leaf of $T_l$, then $P_{l'} + X$ is at least as good a solution and corresponds to a leaf of $T_{l'}$. This justifies our making $l$ inactive.

We apply this algorithm to knapsack problems in which the costs and weights are equal and $B$ is the sum of the weights divided by two and rounded down. Thus, we are considering a generalization of the partition problem., and an optimal solution can have profit at most $B$. Further, $C_S$ is at most $B$, since $\frac{p_i}{w_i} = 1$ for all $i$. Thus, we only apply (B) at a node if the corresponding $d$ exceeds 1, or we find a solution of value $B$. Further we only apply (C) at a node $l$ if there is another node $l'$ such that: $S_l = S_{l'}$, and $\Sigma_{I_j \in P_l} w_j = \Sigma_{I_j \in P_{l'}} w_j$ (Note that by construction if $S_l = S_{l'}$, we must have $P_l \neq P_{l'}$).

We choose a random knapsack instance of this type by choosing each $w_i = p_i$ to be a uniform integer between 1 and $10^{\frac{n}{2}}$, and then setting $B = \lfloor \frac{\sum_{i=1}^{n} w_i}{2} \rfloor$. We prove a theorem of Chvatal, originally proven in [**?**]

**Theorem 11** **Whp** *none of the $2^{\frac{n}{10}}$ nodes in the first $\frac{n}{10}$ layers of the tree are made inactive. Hence,* **whp** *the algorithm takes exponential time.*

**Proof**

**20** **Whp** *the following properties hold:*

*Property 1. There does not exist a set of $\frac{n}{10}$ items the sum of whose weights exceed $B$,*

*Property 2. There do not exist two distinct sets of items with the same weight, whose weights exceed $B$,*

*Property 3. There does not exist a set of items the sum of whose weights is $B$,*

*Property 4. No integer $d$ greater than 1 divides more than $\frac{9n}{10}$ of the items.*

Now, if Property 1 holds then we never apply (A) to a node in the first $\frac{n}{10}$ levels. Similarly, if Properties 3 and 4 hold then we never apply (B) to a node in the first $\frac{n}{10}$ levels. Finally, if Property 2 holds then we never apply (C) to a node in the first $\frac{n}{10}$ levels. So, this result implies the theorem, we leave its proof as an exercise. □

## 6.2 $k$-Median

We have a set $X$ of $n$ points $\{X_1, X_2, \ldots, X_n\}$ with distance $d_{i,j}$ between $X_i$ and $X_j$. The $k$-median problem is to find a set $S \subseteq X$, $|X| = k$ which minimises $\sum_{i=1}^{n} d(X_i, S)$ where $d(X_i, S)$ is the minimum of $d_{i,j}$ over $j \in S$. As an integer program this can be expressed

$$\begin{array}{lrcll} \text{Minimise} & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j}x_{i,j} & & & \\ \text{Subject to} & \sum_{j=1}^{n} x_{i,j} & = & 1 & 1 \le i \le n \\ & \sum_{j=1}^{n} y_j & = & k & \\ & 0 \le x_{i,j} & \le & y_j \le 1 & 1 \le i,j \le n \\ & y_j & \in & \{0,1\} & 1 \le j \le n \end{array}$$

The *strong* linear programming relaxation is obtained by removing the integrality constraint on the $y_j$'s. In practise this has been very useful a linear programming relaxation for branch and bound algorithms. Nevertheless a probabilistic analysis in Ahn, Cooper, Cornuéjols and Frieze [4] shows that in several probabilistic models, including points chosen uniformly in the unit square, the number of branches needed in such a branch and bound algorithm is **whp** at least $n^{\alpha k}$ for some constant $\alpha$, provided $k/\log n \to \infty$ and $k = o((n/\log n)^{1/2})$. Thus in this case a probabilistic analysis does not gel with computational experience.

## 6.3 Quadratic Assignment

Here we have $n$ items which have to be placed in $n$ positions, one item to a position. There is a cost $a_{i,j,p.q}$ associated with placing item $i$ in position $p$ and item $j$ in position $q$. The total cost is the sum of these costs and the problem is to

$$\begin{array}{lrcll} \text{Minimise} & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{n}\sum_{q=1}^{n} a_{i,j,p,q}x_{i,p}x_{j,q} & & & \\ \text{Subject to} & \sum_{p=1}^{n} x_{i,p} & = & 1 & 1 \le i \le n \\ & \sum_{i=1}^{n} x_{i,p} & = & 1 & 1 \le p \le n \\ & x_{i,p} & = & 0/1 & 1 \le i,p \le n \end{array}$$

This is a rather difficult problem and many branch and bound algorithms are based on (i) replacing the terms $x_{i,p}xj,q$ by new 0/1 variables $y_{i,j,p,q}$ and adding suitable linear constraints to make a linear integer program. (ii) Relaxing the integrality of the $y_{i,j,p,q}$ to give a linear program (often this is only done approximately.)

Assume that the $a_{i,j,p,q}$ are independent uniform [0,1] random variables. The expected optimum value then becomes $\approx n^2/2$ – see Section ??. Dyer, Frieze and McDiarmid [36] show that the expected value of the linear relaxation described above is at most $5n + O(1)$ i.e. there is a severe duality gap problem. Not unexpectedly, they go onto show that as a consequence, any branch and bound algorithm based on using the LP relaxation for a bound will **whp** require an exponential number of branches to solve the problem.

## 6.4 Further Results

The first result giving bounds on the average-case complexity of an algorithm are due to Chvatal and concern the maximum stable set problem[**?**]. Further results on this problem are given in Jerrum[**?**] and in Pittelfr.PitNeg. McDiarmid[**?**] obtained difficulty results for vertex colouring. Perhaps the most impressive result of this type concerns the well-known resolution rule for Satisfiability. Chvatal and Szemeredi[**?**] showed that it will take exponential time **whp** for an appropriate probability distribution.

# 7 Non-Algorithmic Issues

The performance of some of our algorithms may be highly sensitive to the probability distribution which we use. We present two examples here, concerning the asymmetric TSP and SAT. We also present results in the opposite direction, which show that for some problems, an algorithm's performance is essentially independent of which input it is given. I.e. we may show that under some probability distributions, the algorithm will get close to the same answer on all but a tiny tiny fraction of the inputs. As examples we consider Bin-Packing and the Quadratic assignment problem.

## 7.1 Thresholds

### 7.1.1 Satisfiability

Given a boolean formula $\omega$ in conjunctive normal form, the *satisfiability problem* (SAT) is to determine whether there is a truth assignment that satisfies $\omega$. Since SAT is NP-complete, one is interested in efficient heuristics that perform well "on average," or with high probability. The choice of the probabilistic space is crucial for the significance of such a study. In particular,

it is easy to decide SAT in probabilistic spaces that generate formulas with large clauses [51]. To circumvent this problem, recent studies have focused on formulas with exactly $k$ literals per clause (the $k$-SAT problem). Of particular interest is the case $k = 3$, since this is the minimal $k$ for which the problem is NP-complete.

Let $V_n$ be a set of $n$ variables. We define a uniform probability space $\Omega_{m,n}^{(k)}$ on the set of all $m = \lfloor cn \rfloor$ clause formulae over the variables which have exactly $k$ literals per clause.

Most practical algorithms for the satisfiability problem (such as the well-known Davis-Putnam algorithm [30]) work iteratively. At each iteration, the algorithm selects a literal and assigns it the value 1. All clauses containing this literal are erased from the formula, and the complement of the chosen literal is erased from the remaining clauses. Algorithms differ in the way they select the literal for each iteration. The following three rules are the most common ones:

1. *The unit clause rule:* If a clause contains only one literal, that literal must have the value 1;
2. *The pure literal rule:* If a formula contains a literal but does not contain its complement, this literal is assigned the value 1;
3. *The smallest clause rule:* Give value 1 to a (random) literal in a (random) smallest clause.

Broder, Frieze and Upfal [18] analysed an algorithm based entirely on the pure literal rule. They showed that when $k = 3$ the pure literal rule alone is sufficient to find, with high probability, a satisfying assignment for a random formula $\omega \in \Omega_{m,n}^{(3)}$, for $c = m/n \leq 1.63$. On the other hand, if $c > 1.7$, then the pure literal rule by itself does not suffice. The gap between 1.63 and 1.7 has been closed independently by Brightwell, Broder, Frieze, Mitzenmacher and Upfal [17] and Molloy and Wormald [78]. In fact if $t$ is the solution to

$$(1-t)^{1/2} + \exp\left(\frac{-1}{2[(1-t)^{-1/2}-1]}\right) - 1 = 0,$$

and

$$c_0 = \frac{1}{3[(1-t)^{1/2} - (1-t)]}$$

then then the pure literal rule is sufficient **whp** when $c < c_0$ and the pure literal rule will almost surely be insufficient when $c > c_0$.

Chao and Franco [23],[24], Chvátal and Reed [26] and Frieze and Suen [48] analysed based on the small clause rule:

**begin**
  **repeat**
    choose a literal $x$;
    remove all clauses from $\omega$ that contain $x$ and remove $\bar{x}$ from any remaining clause;
    if a clause becomes empty - HALT, FAILURE;
  **until** no clauses left;

HALT, SUCCESS
**end**

In particular, in the case of 3-SAT Frieze and Suen showed that if $c_1 \approx 3.003$ is the solution to the equation

$$3c - 2\log c = 6 - 2\log(2/3),$$

then a small clause rule combined with some limited backtracking is enough to find a satisfying assignment **whp** whenever $c < c_1$. From the other end it is easy to show that if $c$ is sufficiently large then then **whp** there is no satisfying assignment. There have been several attempts to estimate how large is large. Kamath, Motwani, Palem and Spirakis [58] showed that 4.758 is large enough for 3-SAT and subsequently Kirousis, Kranakis and Krizanc [65] reduced this to 4.598. Experimental evidence [70, 76] strongly suggests that there exists a threshold $\gamma$, such that formulas are almost surely satisfiable for $c < \gamma$ and almost surely unsatisfiable for $c > \gamma$, where $\gamma$ is about 4.2. This has not been proven rigorously, but such a threshold (namely c=1) is known to exist for 2-CNF formulas [50, 26]. On the other hand, Friedgut [40] has shown that there is a sharp threshold $c_n$ for each $n$. We refer the reader to the paper for an explanation of what this means. Basically, the question now is as to whether $c_n$ tends to a limit as $n \to \infty$.

### 7.1.2 T

he Asymmetric TSP

In this section, we consider the ATSP where each cost is a uniform integer between 0 and $k_n$ for some integer $k_n$. If $k_n < \frac{n}{2\log n}$ then a variant of Karp and Steele's algorithm can be used to show that some optimal AP solution can be patched to an optimal ATSP solution using only zero cost edges. Frieze, Karp and Reed [47] using a more involved argument, showed:

$$ATSP - AP = \begin{cases} 0 & \textbf{whp} & \text{if } L_n/n \to 0 \\ 0 & \text{Positive probability} & \text{if } L_n = cn \\ > 0 & \textbf{whp} & \text{if } L_n/n \to \infty \end{cases}$$

Their work was partially motivated by computational results of Miller and Plekny[**?**].

**Research problem:** Determine the relationship between the optimal solutions for AP and ATSP when $k_n = cn$.

**Research Problem:** Show that for $k_n$ sufficiently large, the Branch and Bound procedure of Miller and Plekny which is based on Karp and Steele's algorithm, takes exponential time **whp**.

## 7.2 Concentration

### 7.2.1 Knapsack

To be filled in by Colin or with a reference to Colin

### 7.2.2 Quadratic Assignment Problem

There are cases where probabilistic analysis can lead to counter-intuitive results which make near optimization a trivial exercise **whp**.

Consider the Quadratic Assignment Problem (QAP) defined in Section ??. As we have seen any branch and bound algorithm based on a natural linear programming relaxation will take exponential time **whp**. On the other hand, we see next that **whp** one cannot avoid finding a solution which is near optimal.

Fix an assignment $\mathbf{x} = (x_{i,j})$ and let

$$Z_{\mathbf{x}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{n}\sum_{q=1}^{n} a_{i,j,p,q}x_{i,p}x_{j,q}.$$

The values $a_{i,j,p,q}$ are independent uniform [0,1]. Hence, for a fixed $\mathbf{x}$, the random variable $Z_{\mathbf{x}}$ has mean

$$\mathbf{E}(Z_{\mathbf{x}}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{n}\sum_{q=1}^{n} x_{i,p}x_{j,q} = \frac{n^2}{2}.$$

$Z_{\mathbf{x}}$ is the sum of $n^2$ independent random variables $(a_{i,j,p,q} : \ x_{i,p} = x_{j,q} = 1)$ and so applying Hoeffding's theorem

$$\mathbf{Pr}(|Z_{\mathbf{x}} - n^2/2| \geq t) \leq e^{-2t^2/n^2}$$

for any $t > 0$. In particular, if $t = \omega n^{3/2}\sqrt{\log n}$ where $\omega = \omega(n) \to \infty$ then we have

$$\mathbf{Pr}(|Z_{\mathbf{x}} - n^2/2| \geq \omega n^{3/2}\sqrt{\log n}) \leq e^{-2\omega^2 n\log n}.$$

Now there are only $n!$ solutions to QAP and so

$$\mathbf{Pr}(\exists \mathbf{x} : \ |Z_{\mathbf{x}} - n^2/2| \geq \omega n^{3/2}\sqrt{\log n}) \leq n!e^{-2\omega^2 n\log n} \to 0.$$

Our conclusion therefore is that **whp** *every* solution to QAP has an objective value in the interval $[n^2/2 - \omega n^{3/2}\sqrt{\log n}, n^2/2 + \omega n^{3/2}\sqrt{\log n}]$ and taking any $\omega = o((n/\log n)^{1/2})$ we see that *any* solution is within $1 + o(1)$ of the optimum.

This was first observed by Burkard and Fincke [21]. More recent examples of this phenomenon are given by Barvinok [8] and Szpankowski [86].

# References


[1] I.Adler, R.M.Karp and R.Shamir, *A family of simplex variants solving an $n \times d$ linear program in $O(\min\{n^2, d^2\})$ expected number of pivot steps*, University of California, Computer Science Division, Berkeley, (1983).

[2] I.Adler and N.Megiddo, *A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension*, Department of Industrial Engineering and Operations Research, University of California, Berkeley, (1983).

[3] I.Adler, N.Megiddo and M. J. Todd, *New results on the average behavior of simplex algorithms* Bulletin of the American Mathematical Society 11 (1984) 378-82.

[4] S.Ahn, C.Cooper, G.Cornuéjols and A.M.Frieze, *Probabilistic analysis of a relaxation for the k-median problem*, Mathematics of Operations Research 13 (1988) 1-31.

[5] N.Alon and N.Kahale, *A spectral technique for coloring random 3-colorable graphs*, Proceedings of the 26th Annual ACM Symposium on Theory of Computing, (1994) 346-355.

[6] J.Aronson, A.M.Frieze and B.G.Pittel, *Maximum matchings in sparse random graphs: Karp-Sipser re-visited*, Random Structures and Algorithms 12 (1998) 111-178.

[7] J.Beardwood, J.H.Halton and J.M.Hammersley, *The shortest path through many points*, Proceedings of the Cambridge Philosophical Society 55 (1959) 299-327.

[8] A.Barvinok, *Measure concentration in optimization*, Mathematical Programming, Series B, 79 (1997) (Lectures on Mathematical Programming, ISMP 97, T.M. Liebling and D. de Werra eds.), 33-53.

[9] C.Berge and Fournier,

[10] C.Blair, *Random linear programs with many variables and few constraints*, Mathematical Programming 34 (1986) 62-71.

[11] P.Bloniarz, *A shortest-path algorithm with expected time $O(n^2 \log n \log^* n)$*, SIAM Journal on Computing 12 (1983) 588-600.

[12] B.Bollobás, Random Graphs, Academic Press, 1984.

[13] B.Bollobás, T.I.Fenner and A.M.Frieze, *An algorithm for finding hamilton paths and cycles in random graphs*, Combinatorica 7 (1987) 327-341.

[14] R.Boppana, *Eigenvalues and graph bisection: an average case analysis*, Proceedings of the 28th Annual IEEE Symposium on the Foundations of Computer Science (1987) 280-285.

[15] K.H.Borgwardt, *The average number of pivot steps required by the simplex method is polynomial*, Zeitschrift für Operations Research 26 (1982) 157-177.

[16] K.H.Borgwardt, *The simplex method, a probabilistic analysis*, Springer-Verlag, 1987.

[17] G.Brightwell, A.Z.Broder. A.M.Frieze, M.Mitzenmacher and E.Upfal, *On the satisfiability and maximum satisfiability of random 3-CNF formulas*, to appear.

[18] A.Z.Broder. A.M.Frieze and E.Upfal, *On the satisfiability and maximum satisfiability of random 3-CNF formulas*, Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (1993) 341-351.

[19] A.Z.Broder, A.M.Frieze, S.Suen and E.Upfal, *Optimal construction of edge disjoint paths in random graphs*, Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (1994) 603-612.

[20] A.Z.Broder, A.M.Frieze, S.Suen and E.Upfal, *Optimal construction of vertex disjoint paths in random graphs*, Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (1996) CHECK.

[21] R.E.Burkard and U.Fincke, *The asymptotic probabilistic behaviour of quadratic sum assignment problems*, Zeitschrift für Operations Research 27 (1983) 73-81.

[22] T.Bui, S.Chaudhuri, T.Leighton and M.Sipser, *Graph bisection with good average case behaviour*, Combinatorica 7 (1987) 171-192.

[23] M.T.Chao and J.Franco, *Probabilistic analysis of two heuristics for the 3-satisfability problem*, SIAM Journal on Computing 15 (1986) 1106-1118.

[24] M.T.Chao and J.Franco, *Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the $k$ satisfiabiable problem*, Information Science 51 (1990) 289-314.

[25] H.Chen and A.M.Frieze, H.Chen and A.M.Frieze, *Coloring Bipartite Hypergraphs*, submitted for publication.

[26] V.Chvátal and B.Reed, *Mick gets his (the odds are on his side)*, Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, (1992) 620-627.

[27] E.G.Coffman and D.S.Johnson, *Approximation algorithms for bin packing: a survey*, in Approximation Algorithms for NP-hard Problems, D.S.Hochbaum (Ed.), PWS 1997.

[28] E.G.Coffman and G.S.Lueker, *Probabilistic analysis of packing and partitioning algorithms*, John Wiley and Sons, New York, 1991.

[29] C.Cooper, A.M.Frieze, K.Mehlhorn and V.Priebe, C.Cooper, A.M.Frieze, K.Mehlhorn and V.Priebe, *Average-case analysis of shortest-paths algorithms in the vertex potential model*, Randomization and approximation techniques in Computer Science (Proceedings of RANDOM '97) Lecture Notes in Computer Science 1269 (1997) 15-26.

[30] M.Davis and H.Putnam, *A computing procedure for quantification theory*, Journal of the ACM 7 (1960) 201-215.

[31] E.Dijkstra, *A note on two problems in connection with graphs*, Numerische Mathematische 1 (1959) 269-271.

[32] M.E.Dyer and A.M.Frieze, *Fast algorithms for some random NP-hard problems*, Journal of Algorithms 10 (1989) 451-489.

[33] M.E.Dyer and A.M.Frieze, *Probabilistic analysis of random m-dimensional knapsack problems*, Mathematics of Operations Research 14 (1989) 162-176.

[34] M.E.Dyer and A.M.Frieze, *On patching algorithms for random asymmetric travelling salesman problems*, Mathematical Programming 46 (1990) 361-378.

[35] M.E.Dyer and A.M.Frieze, *Probabilistic analysis of the generalised assignment problem*, Mathematical Programming 55 (1992) 169-181.

[36] M.E.Dyer, A.M.Frieze and C.J.H.McDiarmid, *Linear programs with random costs*, Mathematical Programming 35 (1986) 3-16.

[37] M.E.Dyer, A.M.Frieze and B.G.Pittel, *On the average performance of the greedy algorithm for finding a matching in a graph*, Annals of Applied Probability 3 (1993) 526-552.

[38] P.Erdos and R.J.Wilson, *On the chromatic index of almost all graphs*, Journal of Combinatorial Theory B 23 (1977) 255-257.

[39] G.Frederickson, *Probabilistic analysis for simple one and two dimensional bin packing algorithms*, Information Processing Letters 11 (1980) 156-161.

[40] E.Friedgut, *Necessary and Sufficient conditions for Sharp Thresholds of Graph Proprties, and the $k$-sat Problem*, to appear.

[41] A.M.Frieze, *An algorithm for finding Hamilton cycles in random digraphs*, Journal of Algorithms 9 (1988) 181-204.

[42] A.M.Frieze, *On the Lagarias-Odlyzko algorithm for the subset-sum problem*, SIAM Journal on Computing 15 (1986) 536-539.

[43] A.M.Frieze and G.R.Grimmett, *The shortest path problem for graphs with random arc-lengths*, Discrete Applied Mathematics 10 (1985) 57-77.

[44] A.M.Frieze, W.Jackson, C.J.H.McDiarmid and B.Reed, *Edge-colouring random graphs*, Journal of Combinatorial Theory B 45 (1988) 135-149.

[45] A.M.Frieze and C.J.H.McDiarmid, *Algorithmic theory of random graphs*, Random Structures and Algorithms 10 (1997) 5-42.

[46] A.M.Frieze, J.Radcliffe and S.Suen, *Analysis of a simple greedy matching algorithm on random cubic graphs*, Combinatorics, Probability and Computing 4 (1995) 47-66.

[47] A.M.Frieze, R.M.Karp and B.A.Reed, *When is the assignment bound asymptotically tight for the asymmetric traveling-salesman problem?*, SIAM Journal on Computing 24, 484-493. *Analysis of a simple greedy matching algorithm on random cubic graphs*, Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (1993) 341-351.

[48] A.M.Frieze and S.Suen, *Analysis of two simple heuristics on a random instance of kSAT*, Journal of Algorithms 20, 312-355.

[49] A.M.Frieze and L.Zhao, *Optimal Construction of Edge-Disjoint Paths in Random Regular Graphs*, in preparation.

[50] A. Goerdt, *A threshold for unsatisfiability*, to appear in *17th International Symposium on Mathematical Foundations of Computer Science*, Prague, Czechoslovakia, August 1992.

[51] A.Goldberg, *Average case complexity of the satisfiability problem*, Proceedings of 4th Workshop on Automated Deduction, (1979) 1-6.

[52] A.V.Goldberg and A.Marchetti-Spaccemela, *On finding the exact solution of a 0,1 knapsack problem*, Proceedings of the 16th Annual ACM Symposium on the Theory of Computing (1984) 359-368.

[53] Y.Gurevich and S.Shelah, *Expected computation time for Hamiltonian path problem*, SIAM Journal on Computing 16 (1987) 486-502.

[54] M.Haimovich, *The simplex algorithm is very good! – on the expected number of pivot steps and related properties of random linear programs*, Columbia University, New York 1983.

[55] M.Held and R.M.Karp, *A Dynamic Programming approach to sequencing problems*, SIAM Journal of Applied Mathematics 10 (1962) 196-210.

[56] D.S.Hochbaum, *An exact sub-linear algorithm for the max-flow, vertex-disjoint paths and communication problems on random graphs*, Operations Research 40 (1992) 923-935.

[57] M.R.Jerrum and Sorkin, *Simulated Annealing for Graph Bisection*, Proceedings of the 34th Annual IEEE Symposium on the Foundations of Computer Science (1993) 94-103.

[58] A.Kamath, R.Motwani, K.Palem and P.Spirakis, *Tail bounds for occupancy and the satisfiability threshold conjecture*, Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, (1994) 592-603.

[59] N.Karmarker and R.M.Karp, *The differencing method of set partitoning*, Technical Report UCB/CSD 82/113 Computer Science Division (EECS), University of California, Berkeley 1982.

[60] N.Karmarker, R.M.Karp, G.S.Lueker and A.M.Odlyzko, *Probabilistic analysis of optimum partitioning*, Journal of Applied Probability 23 (1986) 626-645.

[61] R.M.Karp, *Probabilistic Analysis of Partitioning Algorithms for the Traveling-Salesman Problem in the Plane*, Mathematics of Operations Research 2 (1977) 209-244.

[62] R.M.Karp, *A patching algorithm for the non-symmetric traveling salesman problem*, SIAM Journal on Computing 8 (1979) 561-573.

[63] R.M.Karp and M.Sipser, *Maximum matchings in sparse random graphs*, Proceedings of the 22nd Annual IEEE Symposium on the Foundations of Computer Science (1981) 364-375.

[64] R.M.Karp and J.M.Steele, *Probabilistic analysis of heuristics* in The traveling salesman problem: a guided tour of combinatorial optimization, E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan and D.B.Shmoys Eds. (1985).

[65] L.M.Kirousis, E.Kranakis and D.Krizanc, *Approximating the unsatisfiability threshold of random formulas*, Proceedings of the 4th Annual European Symposium on Algorithms (1996) 27-38.

[66] V.Klee and G.Minty, *How good is the simplex algorithm?*, in Inequalities III, O. Sisha Ed.,Academic Press (1972) 159-175.

[67] D.E.Knuth, R.Motwani and B.G.Pittel, *Stable husbands*, Random Structures and Algorithms 1 (1990) 1-14.

[68] S.G.Kolliopolous and C.Stein, *Finding real-valued single-source shortest paths in $o(n^3)$ expected time*, Proceedings of the 5th Conference on Integer Programming and Combinatorial Optimization (1996) 94-104.

[69] J.C.Lagarias and A.Odlyzko, *Solving low-density subset sum problems*, Journal of ACM 32 (1985) 229-246.

[70] T.Larabee, *Evidence for the satisfiability threshold for random 3CNF formulas*,

[71] M.Luby and P.Ragde, *Bidirectional search is $O(\sqrt{n})$ faster then Dijkstra's shortest path algorithm*, Algorithmica 4 (1989) 551-567.

[72] G.S.Lueker, *On the average distance between the solutions to linear and integer knapsackproblems*, Applied Probability - Computer Science, The Interface 1 (1982) 489-504

[73] J.Mamerand K.Schilling, *On the growth of random knapsacks*, Discrete Applied Mathetics ???

[74] K.Mehlhorn and V.Priebe, *On the all pairs shortest path algorithm of Moffat and Takaoka*, Random Structures Algorithms 10 (1997) 205-220.

[75] Merkle and Hellerman,

[76] D.Mitchell, B.Selman and H.Levesque, Hard and easy distributions of SAT problems, CHECK.

[77] Moffatt and Takaoka, *An all pairs shortest path algorithm with expected time* $O(n^2 \log n)$, Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science (1985) 101-105.

[78] M.Molloy and N.Wormald,

[79] N.Robertson and P.D.Seymour, *Graph minors-XIII: The disjoint paths problem*, to appear.

[80] R.Sedgewick and P.Flajolet, An introduction to the analysis of algorithms, Addison-Wesley, New York, 1996.

[81] E.Shamir and E.Upfal, *A fast construction of disjoint paths in networks*, Annals of Discrete Mathematics 24 (1985) 141-154.

[82] S.Smales, *On the average number of steps of the simplex method of linear programming*, Mathematical Programming 27 (1983) 241-263.

[83] S.Smales, *The problem of the average speed of the simplex method*, in Mathematical Programming: The State of the Art, A.Bachem, M.Grötschel and B.Korte (1983) 530-539.

[84] Spira, *A new algorithm for finding all shortest paths in a graph of positive arcs in average time* $O(n \log n)$, SIAM Journal on Computing 2 (1973) 28-32.

[85] M.J.Steele, Probability theory and combinatorial optimization, CBMS-NSF Regional Conference Series in Applied Mathematics 69, 1997.

[86] W.Szpankowski, *Combinatorial optimization problems for which almost every algorithm is asymptotically optimal*, Optimization 33 (1995) 359-368.

[87] A.Thomason, *A simple linear expected time algorithm for Hamilton cycles*, Discrete Mathematics 75 (1989) 373-379.

[88] Tinhofer, *A probabilistic analysis of some greedy cardinality matching algorithms*, Annals of Operations Research 1 (1984) 239-254.

[89] L.G.Valiant and and G.J. Brebner, *Universal schemes for parallel computation*, Proceedings of the 13th Annual ACM Symposium on Theory of Computing (1981) 263-277.

[90] V.G.Vizing, *On an estimate of the chromatic class of a $p$-graph* (Russian) Diskret. Analiz. 3 (1964) 25-30.

[91] B.Yakir, *The differencing algorithm LDM for partitioning': a proof of a conjecture of Karmarker and Karp*, Mathematics of Operations Research 21 (1996) 85-99.