Chapter 4

Random Walks

A random walk in a graph is a process that moves between the vertices of the graph in a random way. The graph can be described explicitly or implicitly, in the latter case, the random walk might describe a process that changes its state every turn. We start the chapter with two applications where k-SAT for $k = 2$ and $k = 3$ are solved by local search based algorithms that can be analyzed by investigating random walks.

4.1 Applications

We analyze two algorithms whose analysis reduces to analyzing random walks on a line graph. The second application is what we are really interested in, a local search based algorithm for 3-SAT. The algorithm was developed and analyzed by Schöning [Sch99, Sch02] for k-SAT. It is an elegant randomized algorithm that performs surprisingly well. When it was developed, it was the fastest known algorithm for 3-SAT (of its type, i.e. a randomized exact algorithm with a one-sided failure probability). It has a running time of $\mathcal{O}(1.334^n \cdot \text{poly}(n))$. The currently fastest algorithm is a randomized algorithm due to Hertli [Her14] with a running time of $\mathcal{O}(1.30704^n \cdot \text{poly}(n))$. Schönings algorithm was derandomized by Moser and Scheder [MS11]. The best known deterministic algorithm for 3-SAT is a derandomized version of an algorithm by Hofmeister, Schöning, Schuler and Watanabe [HSSW07], done by Makino, Tamaki and Yamamoto [MTY13]. The running time of this algorithm is $\mathcal{O}(1.3302^n \cdot \text{poly}(n))$.

To understand the basic analysis principle, we first look at a similar algorithm for 2-SAT which was previously developed by Papadimitriou [Pap91].

The input to the *k-satisfiability (k-SAT) problem* is a Boolean formula that is the conjunction of m clauses with at most k literals over n variables $x_1, \ldots, x_n$. Recall that a *literal* is either x_i or $\overline{x_i}$ and that a *clause* is the disjunction of at most k literals. For example,

$$(x_1 \vee \overline{x_3}) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge x_4$$

is a valid input to the 2-SAT problem. The k-SAT problem is to decide whether an assignment a of x exists that satisfies the formula. The 2-SAT problem can be solved in polynomial time. For $k \geq 3$, the k-SAT problem is NP-hard [Kar72].

4.1.1 A local search algorithm for 2-SAT

Papadimitriou's 2-SAT algorithm is given a 2-SAT formula ϕ over n variables $x_1, \ldots, x_n$ and computes an assignment a. The algorithm starts with an arbitrary assignment and then adjusts this assignment. In every step, it chooses an arbitrary clause C that is not satisfied. In order to satisfy C, the assignment of at least one of the two involved variables needs to be changed. Instead of deciding for one based on the structure of ϕ, the algorithm chooses uniformly at random. Assume that ϕ is satisfiable and that a^* is an assignment that satisfies all clauses. Then both literals in C have the inverse assignment compared to a^*, since C would otherwise be satisfied. The algorithm chooses one of them uniformly at random and changes the assignment of the corresponding variable. With probability at least 1/2, this increases the number of variables whose assignment agrees with a^* by one (the Hamming distance of a and a^* is reduced by one). With the remaining probability, the number of variables with matching assignments decreases by one (the Hamming distance increases). The goal of the algorithm is to agree with one satisfying assignment in all variables. The important step is now to prove that this happens with high probability when the algorithm does sufficiently many steps. The following pseudo code `Rand-2-SAT(`ϕ, t, a'`)` realizes Papadimitriou's algorithm. Here, ϕ is the input formula, t is a parameter that controls the number of steps and a' is a parameter that allows us to start the algorithm with an arbitrary assignment.

```
Rand-2-SAT($\phi, t, a'$)
  1. initialize the assignment $a$ with $a'$
  2. for $i = 1$ to $t$ do
  3.    if $a$ does not satisfy $\phi$ then
  4.       choose an arbitrary clause $C$ in $\phi$ that is not satisfied by $a$
  5.       choose a literal in $C$ uniformly at random and change its assignment
  6. return "YES" if $x$ satisfies $\phi$, else return "NO"
```

The algorithm has one-sided error because it only returns `YES` if it finds a satisfying assignment. If ϕ is not satisfiable, then it cannot find such an assignment and outputs `NO`, which is correct in this case. If ϕ is satisfiable, then the algorithm is correct if it finds a satisfying assignment, otherwise, it errs.

To analyze the failure probability of `Rand-2-SAT` in the case that ϕ is satisfiable, we interpret the execution as a random walk. The graph is implicitly defined. It consists of a node i for all $i \in \{0, \ldots, n\}$, and node i represents the state that the current assignment agrees with an (arbitrary but fixed) satisfying assignment a^* in exactly i variables. If n is reached, then the algorithm correctly outputs `YES`. If n is not reached, it might still output `YES` because there could be other satisfying assignments. For formulas with a unique satisfying assignment, the algorithm will err if the random walk does not reach n. We thus analyze the probability that the algorithm reaches n within t steps which is a lower bound on the success probability of `Rand-2-SAT`.

$$\begin{tikzcd}
0 \arrow[r, bend left, "1"] & 1 \arrow[l, bend left, "\leq 1/2"] \arrow[r, bend left, "\geq 1/2"] & 2 \arrow[l, bend left, "\leq 1/2"] \arrow[r, bend left, "\geq 1/2"] & 3 \arrow[l, bend left, "\leq 1/2"] \arrow[r, bend left, "\geq 1/2"] & 4 \arrow[l, bend left, "\leq 1/2"] \arrow[r, bend left, "\geq 1/2"] & 5 \arrow[l, bend left, "\leq 1/2"] \arrow[r, bend left, "\geq 1/2"] & \cdots \arrow[l, bend left, "\leq 1/2"] \arrow[r, bend left, "\geq 1/2"] & n-1 \arrow[l, bend left, "\leq 1/2"] \arrow[r, bend left, "\geq 1/2"] & n \arrow[loop right, "1"]
\end{tikzcd}$$

The initial assignment is arbitrary, so the algorithm starts at an arbitrary node i. If the state is 0, then flipping any variable increases the number of agreeing variables by one. Thus, going from 0 to 1 has probability 1. If the state is n, then the algorithm has found a^*, outputs `YES` and ends. Thus, once n is reached, it is never left. For all other $i \in \{1, \dots, n-1\}$, the algorithm chooses a clause C that is not satisfied. At least one of the two involved variables is in disagreement with a^*. It is also possible that both variables are assigned differently than in a^*. Thus, the probability to go to $i+1$ is either 1/2 or 1 (i.e. at least 1/2), and the probability to go to $i-1$ is either 1/2 or 0 (i.e. at most 1/2).

Lemma 4.1. *Let ϕ be a 2-SAT formula that is satisfiable. Let a' be an arbitrary assignment. Let Z be the number of steps until **Rand**-2-SAT(ϕ, t, a') (run with sufficiently large t) finds a satisfying assignment. Then $\mathbf{E}[Z] \leq n^2$.*

Proof. Let a^* be an arbitrary but fixed satisfying assignment for ϕ. We analyze the expected number of steps until `Rand-2-SAT(`ϕ, t, a'`)` finds a^* under the assumption that it never finds a satisfying assignment that is not a^*. This number is an upper bound for the expected number of steps until the algorithm finds any satisfying assignment.

Based on a^*, we consider the random walk as discussed above. For all $j \in \{0, \dots, n\}$, let Z_j be the number of steps that `Rand-2-SAT(`ϕ, t`)` needs to reach state n from state j. Thus, Z_j is random variable. Observe that $Z_j \geq Z_{j'}$ for $j \leq j'$, that $Z_0 = 1 + Z_1$ and that $Z_n = 0$. For $j \in \{1, \dots, n-1\}$, we argued above that $Z_j = 1 + Z_{j-1}$ with probability at least 1/2 and $Z_j = 1 + Z_{j+1}$ with probability at most 1/2.

We define $h_j = \mathbf{E}[Z_j]$ as the expected number of steps needed to reach n from j. Observe that proving $\max_{j=0,\dots,n} h_j \leq n^2$ proves the lemma. It is $h_n = 0$ because Z_n is always zero. Furthermore, $h_0 = \mathbf{E}[Z_0] = \mathbf{E}[1 + Z_1] = 1 + \mathbf{E}[Z_1] = 1 + h_1$. For $j \in \{1, \dots, n-1\}$, we get the recurrence relation

$$h_j = \mathbf{E}[Z_j] \leq \frac{1}{2}(1 + \mathbf{E}[Z_{j-1}]) + \frac{1}{2}(1 + \mathbf{E}[Z_{j+1}]) = 1 + \frac{1}{2} \cdot h_{j-1} + \frac{1}{2} h_{j+1}.$$

(Remark: Since $Z_j \geq Z_{j'}$ for $j \leq j'$, we in particular have $Z_{j-1} \geq Z_{j+1}$). This means that the term $p(1+\mathbf{E}[Z_{j-1}])+(1-p)(1+\mathbf{E}[Z_{j+1}])$ for $p \leq 1/2$ is maximized for $p = 1/2$. Thus, the first inequality holds). By induction, we show that for all $j \in \{0, \dots, n-1\}$ it holds that

$$h_j \leq h_{j+1} + 2j + 1. \tag{4.1}$$

For $j = 0$, it is $h_0 = h_1 + 1 = h_1 + 2 \cdot 0 + 1$. For $j \in \{1, \ldots, n-1\}$, we compute

$$\begin{aligned} h_j \le 1 + \frac{1}{2} \cdot h_{j-1} + \frac{1}{2} \cdot h_{j+1} &\overset{I.H.}{\le} 1 + \frac{1}{2}(h_j + 2(j-1) + 1) + \frac{1}{2} \cdot h_{j+1} \\ &= 1 + \frac{1}{2}h_j + (j-1) + \frac{1}{2} + \frac{1}{2} \cdot h_{j+1}. \end{aligned}$$

We transform the inequality to obtain that

$$\begin{aligned} & & h_j &\le 1 + \frac{1}{2}h_j + (j-1) + \frac{1}{2} + \frac{1}{2} \cdot h_{j+1} \\ \Leftrightarrow & & \frac{1}{2}h_j &\le 1 + (j-1) + \frac{1}{2} + \frac{1}{2} \cdot h_{j+1} \\ \Leftrightarrow & & h_j &\le 2(j-1) + 3 + h_{j+1} \\ \Leftrightarrow & & h_j &\le 2j + 1 + h_{j+1} \end{aligned}$$

which proves inequality (4.1). This implies that

$$\begin{aligned} h_j = h_{j+1} + 2j + 1 &= h_{j+2} + 2(j+1) + 1 + 2j + 1 \\ &= \ldots = h_n + 2(n-1) + 1 + \ldots + 2j + 1 \\ &= \sum_{k=j}^{n-1}(2k+1) \le \sum_{k=0}^{n-1}(2k+1) = n + n(n-1) = n^2. \end{aligned}$$

Thus $h_j \le n^2$ for all $j \in \{0, \ldots, n-1\}$ which proves the lemma. □

Lemma 4.1 says that the local search needs n^2 steps on expectation to find a satisfying assignment. We can ensure that a satisfying assignment is found with probability $1-\delta$ by running the algorithm for a sufficiently larger number of steps.

Theorem 4.2. *If* `Rand-2-SAT(`ϕ, t, a'`)` *is called with a satisfiable 2-SAT formula ϕ, a number $t \ge 2n^2\lceil \log_2(1/\delta) \rceil$ and an arbitrary starting assignment a', then it finds a satisfying assignment with probability at least $1-\delta$.*

Proof. We split the t steps into $t' = \lceil \log_2(1/\delta) \rceil$ phases of length $2n^2$. Let A_i be the event that phase i fails, i.e. no satisfying assignment is found. The algorithm fails if all events A_i occur for $i \in \{1, \ldots, t'\}$. The probability that this happens is

$$\mathbf{Pr}(A_1 \cap A_2 \cap \ldots A_{t'}) = \prod_{i=1}^{t'} \mathbf{Pr}(A_i \mid A_{i-1} \cap \ldots \cap A_1)$$

which follows from the definition of conditional probability (compare our derivation on page 14 in Section 1.2). The condition $A_{i-1} \cap \ldots A_1$ only means that phase i starts in an assignment $a'' \ne a^*$. Observe that phase i starting at assignment a'' is identical to calling `Rand-3-SAT(`$\phi, 2n^2, a''$`)`. Lemma 4.1 holds for *any* starting assignment, thus the expected number of steps until an satisfying assignment is found in phase i is n^2. By Markov's inequality, the probability that no satisfying assignment is found within the $2n^2$ steps that the phase is long is bounded by $1/2$. We get that

$$\mathbf{Pr}(A_1 \cap A_2 \cap \ldots A_{t'}) \le \left(\frac{1}{2}\right)^{t'} = \frac{1}{2^{\lceil \log_2(1/\delta) \rceil}} \le \delta.$$

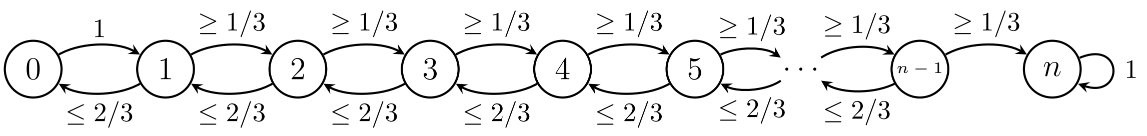
Thus, the probability that the algorithm fails after $2t'n^2$ steps is at most δ. □

4.1.2 Local search algorithms for 3-SAT

Schönings algorithm for k-SAT is *not* identical to the 2-SAT algorithm we saw above. It has an additional spin. Before discussing it (for $k = 3$), we analyze the straightforward generalization of `Rand-2-SAT` to 3-SAT. The resulting algorithm `Rand-3-SAT(`ϕ, t, a'`)` again has the parameters ϕ for the input formula, t for the number of steps and a' for the initial assignment.

```
Rand-3-SAT($\phi, t, a'$)
  1. initialize the assignment $a$ with $a'$
  2. for $i = 1$ to $t$ do
  3.   if $a$ does not satisfy all clauses then
  4.     choose an arbitrary clause $C$ that is not satisfied by $a$
  5.     choose a literal in $C$ uniformly at random and change its assignment
  6. return "YES" if $a$ satisfies all clauses, else return "NO"
```

The analysis of `Rand-3-SAT` for a satisfying instance ϕ differs from the analysis of the algorithm `Rand-3-SAT` in one important aspect. The clauses in ϕ now have up to three literals. For a not satisfied clause C and some fixed satisfying assignment a^*, it is now possible that two of the involved variables agree with a^* and the clause is still not satisfied because a and a^* disagree on the third involved variable. By choosing one of the involved variables uniformly at random, the probability to increase the number of agreeing variables by one can be 1/3, 2/3 or 1. Thus we are only guaranteed that the probability is at least 1/3 and that the probability to decrease the number of agreeing variables is at most 2/3. As for 2-SAT, we model the process by a random walk on a line graph where vertex i represents the state that a and a^* agree in i variables. As before, being in vertex 0 means that the next change increases the number of agreeing variables with probability 1, and being in vertex n means that the assignment is no longer changed. For all $j \in \{1, \ldots, n-1$, we just argued that we go to $j-1$ with probability at most 2/3 and to j with probability at least 1/3.



Intuitively, we expect that the random walk is likely to go into the wrong direction. However, since 3-SAT is NP-hard, we do not expect a polynomial algorithm. Even waiting an exponential number of steps for a satisfying assignment can lead to an algorithm that is good in our eyes. We apply the analytic technique we used in Lemma 4.1 to `Rand-3-SAT`.

Lemma 4.3. *Let ϕ be a 3-SAT formula that is satisfiable and let a^* be an arbitrary satisfying assignment. Let a' be an assignment that agrees with a^* in i variables. Let Z be the number of steps until* `Rand-3-SAT(`ϕ, t, a'`)` *(run with sufficiently large t) finds a satisfying assignment. Then $\mathbf{E}[Z] \le 2^{n+2} - 2^{i+2} - 3(n-i)$.*

Proof. Observe that the lemma assumes that the random walk is in vertex i when we start. Let Z_j be the number of steps that `Rand-3-SAT(`ϕ, t, a'`)` needs to reach state n from state j for all $j \in \{0, \dots, n\}$. Let $h_j = \mathbf{E}[Z_i]$ be the expected number of steps needed to reach n from j. We are required to bound $h_i = \mathbf{E}[Z_i]$.

Again, $Z_j \geq Z_{j'}$ for $j \leq j'$, $Z_0 = 1 + Z_1$ and $Z_n = 0$ hold. For $j \in \{1, \dots, n-1\}$, $Z_j = 1 + Z_{j-1}$ with probability at least $2/3$ and $Z_j = 1 + Z_{j+1}$ with probability at most $1/3$. It is $h_n = 0$ because Z_n is always zero. Furthermore, $h_0 = \mathbf{E}[Z_0] = \mathbf{E}[1 + Z_1] = 1 + \mathbf{E}[Z_1] = 1 + h_1$. For $j \in \{1, \dots, n-1\}$, we get the recurrence relation

$$h_j = \mathbf{E}[Z_j] \leq \frac{2}{3}(1 + \mathbf{E}[Z_{j-1}]) + \frac{1}{3}(1 + \mathbf{E}[Z_{j+1}]) = 1 + \frac{2}{3} \cdot h_{j-1} + \frac{1}{3} \cdot h_{j+1}.$$

This implies that $h_j \leq 2^{n+2} - 2^{j+2} - 3(n-j)$ for all $j \in \{0, \dots, n\}$ (exercise). The lemma assumes that the algorithm starts in i, so the expected number of steps needed until n is reached is bounded by $2^{n+2} - 2^{i+2} - 3(n-i)$. □