輪講資料

2018/10/04

B4 林谷哲郎

1 Introduction

1.1 あらすじ

Imagine that you are an exceptionally tech-savvy security guard of a bar in an undisclosed small town on the west coast of Norway. Every Friday, half of the inhabitants of the town go out, and the bar you work at is well known for its nightly brawls. This of course results in an excessive amount of work for you; having to throw out intoxicated guests is tedious and rather unpleasant labor.

あなたがノルウェーの西海岸の小さな町にあるバーの、非常に技術に精通した警備員であると想像してください。毎週金曜日、街の住人の半分が出て行き、そして、あなたが働いているバーは夜の喧嘩でよく知られています。これらはもちろん、あなたに過度な仕事量をもたらします。酔っぱらいの相手をするのは退屈で、むしろ不愉快な労働です。

So you wish to plan ahead, and only admit people if they will not be fighting with anyone else at the bar. At the same time, the management wants to maximize profit and is not too happy if you on any given night reject more than k people at the door.

そこで、あなたは前もって対策をとることにしました。町は小さいので、あなたは町の住人を全員知っています。バーに入場すると、誰と誰が戦う可能性が高いかもわかります。だから事前に対策を取りたいと思っていて、バーの中で誰とも争わない人だけを入場させたいです。同時に、経営陣は利益を最大限に増やしたいと思っていますし、あなたが k 人より多いゲストを拒めば、あまり喜ばしく思いません。

Therefore, there are the following optimization problems.

したがって、あなたには次のような最適化問題が与えられます。(！)

1.2 問題提起

You have a listof all of the n people who will come to the bar, and for each pair of people a prediction of whether or not they will fight if they both are admitted. You need to know if anyone other than many trouble makers can admit it.
Ensure that fight does not occur between recognized guests.
Let's call this problem a bar fire prevention problem. In case Figure 1.1 shows an example of problem and solution of $k = 3$. You can easily check that this instance has no solution with $k = 2$.

あなたはバーに来るすべての人のリストと、「それぞれの人に対し、誰と喧嘩をするか」の予測があります。あなたはトラブルメーカー以外の多くの人を入店させたいです。今回は、ゲスト間で喧嘩が発生しないようにしたいです。

この問題を Bar Fight Prevention 問題としましょう。図 **1.1** に、$k = 3$ の問題と解の例を示しています。このインスタンスには $k = 2$ の解がないことを簡単にチェックできます。

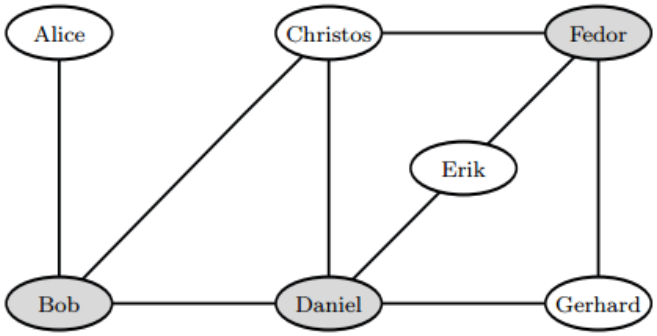

図 **1.1** Bar Fight Prevention 問題の解答例（$k = 3$）。2 人のゲスト間のエッジは、お互いが入店した場合、戦うことを意味する。

1.3 Efficient algorithms for Bar Fight Prevention

Unfortunately, Bar Fight Prevention is a classic NP-complete problem (the reader might have heard of it under the name Vertex Cover), and so the best way to solve the problem is by trying all possibilities, right?
If there are $n = 1000$ people planning to come to the bar, then you can quickly code up the brute-force solution that tries each of the ($2^{1000} \approx 1.07 \cdot 10^{301}$) possibilities.
Sadly, this program won't terminate before the guests arrive, probably not even before the universe implodes on itself.

あいにく、Bar Fight Prevention 問題 は古典的な NP-comp 問題です。(読者は Vertex Cover という名前で聞いたことがあるかもしれません。)[*1]とすると、問題を解決する最善の方法は、すべての可能性を試すことですよね？

バーに来る人が $n = 1000$ 人居たとすれば、($2^{1000} \approx 1.07 \cdot 10^{301}$) の可能性を試す、Brute-Force 法をすばやく書き上げることができます。

悲しいことに、このプログラムはゲストが到着する前に終了しません。おそらく宇宙がそれ自身に激突する前でさえないです。[*2]

Luckily, the number k of guests that should be rejected is not that large,$k \leq 10$.
So now the program only needs to try $\binom{1000}{10} \approx 2.63 \cdot\ 10^{23}$ possibilities. This is much better, but still quite infeasible to do in one day, even with access to supercomputers.
So should you give up at this point, and resign yourself to throwing guests out after the fights break out? Well, at least you can easily identify some peaceful souls to accept, and some troublemakers you need to refuse at the door for sure.

幸いにも、断るべきゲストの数は $k \leq 10$ とそれほど大きくないです。

だから、プログラムは $\binom{1000}{10} \approx 2.63 \cdot\ 10^{23}$ の可能性だけを実行する必要があります。これは遥かに優れていますが、スーパーコンピュータにアクセスしても、一日で計算するのは、まだかなり難しいです。

というわけなので、この時点であなたは諦めるべきですか？衝突が起こった後、ゲストを見捨て、放棄しますか？少なくとも、「あなたは受け入れられる平穏な精神」と「確実に拒否する必要のある厄介者」を簡単に識別できます。

[*1] k 個の頂点を選び、すべての「辺」を被覆する → k 人選び、すべての「衝突」を被覆

[*2] ビッグクランチ（Big Crunch）とは、予測される宇宙の終焉の一形態である。現在考えられている宇宙モデルでは、宇宙はビッグバンによって膨張を開始したとされているが、宇宙全体に含まれる質量（エネルギー）がある値よりも大きい場合には、自身の持つ重力によっていずれ膨張から収縮に転じ、宇宙にある全ての物質と時空は無次元の特異点に収束すると考えられる。（Wiki 調べ）

1.4 考察 その 1

On the other hand, if some guy will fight with at least $k+1$ other guests you have to reject him - as otherwise you will have to reject all of his $k+1$ opponents, thereby upsetting the management. If you identify such a troublemaker (in the example of Fig 1.1, Daniel is such a troublemaker), you immediately strike him from the guest list, and decrease the number k of people you can reject by one.

一方で、もしある男が少なくとも $k+1$ 人の他のゲストと喧嘩するつもりなら、あなたは彼を拒否する必要があります。さもなければ、その人の $k+1$ 人の対戦相手も全て拒否しなければならなくなり、それによって、経営陣が困ってしまいます。

そのようなトラブルメーカーを特定した場合（図 **1.1** の例では、ダニエルはそのような厄介者です）、すぐさまゲストリストから消し去り、ゲストリストから拒否できる人数 k を減らします。（削除した分だけ k が減少）

If there is no one left to strike out in this manner, then we know that each guest will fight with at most k other guests. Thus, rejecting any single guest will resolve at most k potential conflicts. And so, if there are more than k^2 potential conflicts, you know that there is no way to ensure a peaceful night at the bar by rejecting only k guests at the door.

このように打ち切る人が居なくなれば、各ゲストは最大で k 人の他のゲストと喧嘩することになります。

したがって、単一のゲストを拒否すると、最大で k 個の潜在的な衝突が解決されます。

そして、k^2 より大きい潜在的な衝突がある場合、「k 人のゲストのみを拒否することで、バーでの平和な夜を確実にする方法」は無いことがわかります。

As each guest who has not yet been moved to the accept or reject list participates in at least *one* and at most k potential conflicts, and there are at most k^2 potential conflicts, there are at most $2k^2$ guests whose fate is yet undecided.
Trying all possibilities for these will need approximately $\binom{2k^2}{k} \leq \binom{200}{10} \approx 2.24 \cdot 10^{16}$ checks, which is feasible to do in less than a day on a modern supercomputer, but quite hopeless on a laptop.

まだ許可リストまたは拒否リストに移動されていないゲストは、少なくとも 1 個、最大でも k 個の潜在的な衝突に参加するので、（上に書いたように）多くとも k^2 の潜在的な衝突があり、運命がまだ決まっていないゲストは最大で $2k^2$ 人居ます。

これらの可能性をすべて試すには、$\binom{2k^2}{k} \leq \binom{200}{10} \approx 2.24 \cdot 10^{16}$ のチェックが必要であり、現在のスーパーコンピュータでは 1 日もかからないうちに実行可能ですが、ラップトップでは非常に絶望的です。

1.5 考察 その２

If it is safe to admit anyone who does not participate in any potential conflict, what about those who participate in exactly one?
If Alice has a conflict with Bob, but with no one else, then it is always a good idea to admit Alice. Indeed, you cannot accept both Alice and Bob, and admitting Alice cannot be any worse than admitting Bob: if Bob is in the bar, then Alice has to be rejected for sure and potentially some other guests as well.
Therefore, it is safe to accept Alice, reject Bob, and decrease k by one in this case.

衝突がないゲストを入店させることが安全であれば、ひとつだけ衝突がある人はどうですか。

ここではアリスがボブと衝突していますが、ほかの誰も居ない場合、アリスを受け入れることは常に良い考えです。確かに、あなたはアリスとボブの両方を受け入れることはできませんし、アリスを入店させることが、ボブを入店させるより悪いということはないです。（ボブがバーに居ると、アリスは確実に拒否しなければならず、他に衝突している人も拒否しなければなりません。）

したがって、アリスを受け入れ、ボブを拒否し、 k を 1 減らすのは安全です。

This way, you can always decide the fate of any guest with only one potential conflict. At this point, each guest you have not yet moved to the accept or reject list participates in at least *two* and at most k potential conflicts.
It is easy to see that with this assumption, having at most k^2 unresolved conflicts implies that there are only at most k^2 guests whose fate is yet undecided, instead of the previous upper bound of $2k^2$. Trying all possibilities for which of those to refuse at the door requires $\binom{k^2}{k} \leq \binom{100}{10} \approx 1.73 \cdot 10^{13}$ checks.

こうすることで、潜在的な衝突が一つしかないゲストの運命をいつでも決めることができます。この時点でまだ許可リストまたは拒否リストに移動していないゲストは、少なくとも 2 個、最大で k 個の潜在的な衝突があります。

この仮定を用いると、「最大で k^2 個の未解決の衝突を持つことが、運命が決まらないゲストが、$2k^2$ の上限の代わりに、k^2 人のみになる。」ということが容易にわかります。

拒否する人の全ての組み合わせを試すには、$\binom{k^2}{k} \leq \binom{100}{10} \approx 1.73 \cdot 10^{13}$ 回チェックする必要があります。

With a clever implementation, this takes less than half a day on a laptop, so if you start the program in the morning you' ll know who to refuse at the door by the time the bar opens. Therefore, instead of using brute force to go through an enormous search space, we used simple observations to reduce the search space to a manageable size.
This algorithmic technique, using reduction rules to decrease the size of the instance, is called kernelization, and will be the subject of Chapter 2 (with some more advanced examples appearing in Chapter 9)

賢明な実装ではラップトップで半日もかかりませんが、あなたが午前中にプログラムを実行し始めるならば、バーが開くまでに誰を拒否するかがわかるでしょう。

したがって、巨大な探索空間を走査するために Brute-Force 法を使用するのではなく、単純な観測事実から、探索空間を扱いやすいサイズまで減らしました。

インスタンスのサイズを減らすために縮小ルールを利用する、アルゴリズム的な手法をカーネル化といい、Chap 2 の主題になります。（より高度な例は Chap 9 から）

1.6 考察 その３

It turns out that a simple observation yields an even faster algorithm for Bar Fight Prevention. The crucial point is that every conflict has to be resolved, and that the only way to resolve a conflict is to refuse at least one of the two participants. Thus, as long as there is at least one unresolved conflict, say between Alice and Bob, we proceed as follows.

単純な観測事実から、 Bar Fight Prevention のアルゴリズムは更に高速になります。重要な点は、あらゆる衝突が解決されなければならず、衝突を解決する唯一の方法は、2 人のうち少なくとも 1 人を拒否することです。したがって、少なくとも 1 つの未解決な衝突、言い換えればアリスとボブのような関係に対し、つぎのような操作をします。

Try moving Alice to the reject list and run the algorithm recursively to check whether the remaining conflicts can be resolved by rejecting at most $k-1$ guests.
If this succeeds you already have a solution. If it fails, then move Alice back onto the undecided list, move Bob to the reject list and run the algorithm recursively to check whether the remaining conflicts can be resolved by rejecting at most $k-1$ additional guests (see Fig. 1.2).
If this recursive call also fails to find a solution, then you can be sure that there is no way to avoid a fight by rejecting at most k guests

アリスを拒否リストに移動し、最大 $k-1$ 人のゲストを拒否し、残りの衝突を解決できるかどうか確認するために、再帰的にアルゴリズムを実行してください。

これが成功したら、あなたは解決策を手に入れたも同然です。それが失敗したら、アリスを未決定リストに戻し、ボブを拒否リストに移動し、最大で $k-1$ 人のゲストを拒否することによって、残りの衝突を解決できるかどうかを確認するために、再帰的にアルゴリズムを実行してください（図 **1.2** 参照）。

この再帰呼び出しでも解が見つからない場合は、「最大で k 人を拒否することによって、喧嘩を回避する方法」が無いということがわかります。

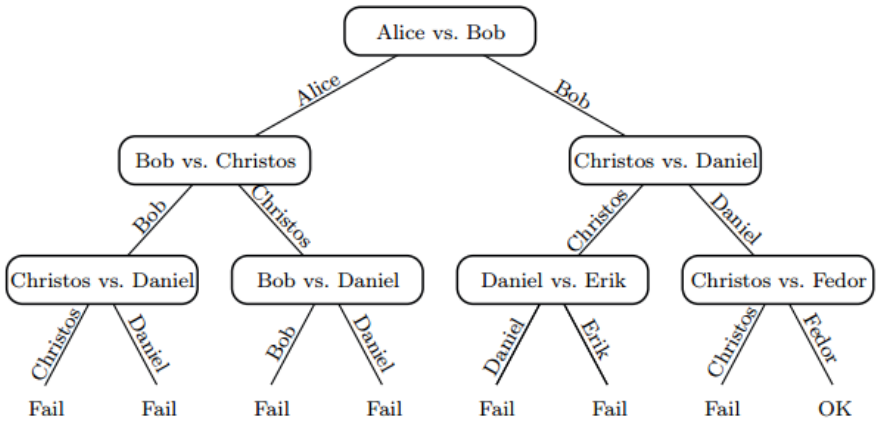

図 **1.2**　$k=3$ の Bar Fight Prevention の探索木。Fail とマークされた葉ではパラメータ k は 0 になりますが、まだ解決されていない衝突があります。探索木の一番右側は答えです。Bob、Daniel、Fedor を拒否した後、これ以上の衝突はありません。

What is the running time of this algorithm? All it does is to check whether all conflicts have been resolved, and if not, it makes two recursive calls.
In both of the recursive calls the value of k decreases by 1, and when k reaches 0 all the algorithm has to do is to check whether there are any unresolved conflicts left.
Hence there is a total of 2^k recursive calls, and it is easy to implement each recursive call to run in linear time $O(n+m)$, where mis the total number of possible conflicts.

このアルゴリズムの実行時間はいくらでしょうか。

すべての衝突が解決されたかどうかを確認するだけで、2 つの再帰呼び出しが行われます。再帰呼び出しの両方において、k の値は 1 だけ減少し、k が 0 になったときに、解決されていない衝突が残っていないかどうかをチェックするアルゴリズムが実行されます。

したがって、合計 2^k の再帰呼び出しがあり、各再帰呼び出しで線形時間 $O(n+m)$ を要することが簡単にわかります。

Let us recall that we already achieved the situation where every undecided guest has at mostkconflicts with other guests, so $m \leq nk/2$.
Hence the total number of operations is approximately $2^k \cdot n \cdot k \leq 2^{10} \cdot 10,000 = 10,240,000$, which takes a fraction of a secondon today's laptops.
Or cell phones, for that matter. You can now make the Bar Fight Prevention app, and celebrate with a root beer.
This simple algorithm is an example of another algorithmic paradigm: the technique of bounded search trees. In Chapter 3, we will see several applications of this technique to various problems

私達は、未確定のゲストが、他のゲストと最大限衝突を起こしている状況 ($m \leq nk/2$) をすでに達成したことを思い出してください。したがって、操作の総数は $2^k \cdot n \cdot k \leq 2^{10} \cdot 10,000 = 10,240,000$ であり、今日のラップトップのほんの数分の一秒を必要とします。

もしくは、携帯電話、いっそついでに Bar Fight Prevention アプリを作って、ルートビア[*3]でお祝いすることができます。

この単純なアルゴリズムは、別のアルゴリズムパラダイム（有界探索木）の例です。Chap 3 では、このテクニックを様々な問題に適用した例をいくつか見ていきます。

[*3] サロンパス風味のジュース、何故ここで。

1.7 まとめ

The algorithm above runs in time $O(2^k \cdot k \cdot n)$, while the naive algorithm that tries every possible subset of k people to reject runs in time $O(n^k)$.
Observe that if k is considered to be a constant (say $k = 10$), then both algorithms run in polynomial time. However, as we have seen, there is a quite dramatic difference between the running times of the two algorithms.
The reason is that even though the naive algorithm is a polynomial-time algorithm for every fixed value of k, the exponent of the polynomial depends on k.
On the other hand, the final algorithm we designed runs in linear time for every fixed value of k! This difference is what parameterized algorithms and complexity is all about.
In the $O(2^k \cdot k \cdot n)$-time algorithm, the combinatorial explosion is restricted to the parameterk: the running time is exponential in k, but depends only polynomially (actually, linearly) on n.
Our goal is to find algorithms of this form.

k 人の全ての可能な部分集合列挙する、単純なアルゴリズムが時間 $O(n^k)$ で実行されている間に、上記のアルゴリズムは時間 $O(2^k \cdot k \cdot n)$ で実行されます。

k が（$k = 10$ のような）定数であると考えられる場合、両方のアルゴリズムは多項式時間で実行されることに注意してください。しかし、これまで見てきたように、2 つのアルゴリズムの実行時間には劇的に異なります。その理由は単純なアルゴリズムが k の全ての固定値に対する多項式時間アルゴリズムであっても、多項式の指数は k に依存するからです。

一方私達が設計した最終的なアルゴリズムは k の固定値ごとに線形時間で実行されます！この相違点はパラメータ化されたアルゴリズムと複雑さの全てです。$O(2^k \cdot k \cdot n)$ 時間アルゴリズムでは、組み合わせ爆発はパラメータ k に制限されています。実行時間は k で指数関数的ですが、n は多項式（実際には線形）にのみ依存します。

私達の目標は、このフォームのアルゴリズムを見つけることです。

1.8 実行時間が $f(k) \cdot n^c$ であるようなアルゴリズム

Algorithms with running time $f(k) \cdot n^c$,for a constant c independent of both n and k, are called *fixed-parameter algorithms*,or FPT algorithms.
Typically the goal in parameterized algorithmics is to design FPT algorithms, trying to make both the $f(k)$ factor and the constant c in the bound on the running time as small as possible.
FPT algorithms can be put in contrast with less efficient XP algorithms for (*slice-wise polynomial*), where the running time is of the form $f(k) \cdot n^{g(k)}$, for some functions f,g. There is a tremendous difference in the running times $f(k) \cdot n^{g(k)}$ and $f(k) \cdot n^c$.
In parameterized algorithmics, k is simply a *relevant secondary measurement* that encapsulates some aspect of the input instance, be it the size of the solution sought after, or a number describing how "structured" the input instance is.

実行時間が $f(k) \cdot n^c$ であるようなアルゴリズムのように、定数 c が n と k の両方から独立している場合、*fixed-parameter algorithms*、もしくは FPT アルゴリズムと呼ばれます。[*4]

$f(k)$ の因子と定数 c の両方を、実行時間の境界内で、できるだけ小さくしようします。[*5]

FPT アルゴリズムは、実行時間がいくつかの関数 f、g で $f(x) \cdot n^{g(k)}$ と表される効率の悪い XP アルゴリズム（スライスワイズ多項式の場合）とは、大きな違いがあります。$f(x) \cdot n^{g(k)}$ と $f(k) \cdot n^c$ では、実行時間に大きな違いがあります。

パラメータ化されたアルゴリズムでは、k は入力インスタンスのいくつかの側面をカプセル化する関連 2 次測定値 (*relevant secondary measurement*) です。それは求められる解のサイズか、入力インスタンスがどのように「構造化」されているかを表す数値です。

[*4] 入力サイズとは独立なパラメータに関してのみ指数時間のアルゴリズム。BAR FIGHT PREVENTION では、n 人与えられ、k 人を選択する。

[*5] BAR FIGHT PREVENTION では、$O(n^k)$ が $O(2^k \cdot k \cdot n)$ に改善された。