ssdnm
środowiskowe
studia doktoranckie
z nauk matematycznych

Dariusz Leniowski

Uniwersytet Warszawski

Aspects of adwords problem – examples

Praca semestralna nr 3

(semestr letni 2012/13)

Opiekun pracy: Piotr Sankowski

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Aspects of adwords problem — examples

Dariusz Leniowski

September 2013

Abstract

This is a follow-up paper to "On edge usage in adwords problem" article by the same author [5]. It provides the most important examples related to rank-minimizing algorithms for one-side online maximum cardinality bipartite matching problem, in particular one that proves the bound of Theorem 3.1 of [5] is tight. This is a part of joint work with Bartłomiej Bosek, Piotr Sankowski and Anna Zych.

1 Introduction

The family of matching problems is a class of highly related problems where a new algorithm may lead to improvements in more general settings. It happened, for example in case of the Hungarian algorithm for assignment problem [4], the Edmonds algorithm for matchings in general graphs [2], or Hopcroft-Karp algorithm for matchings in bipartite graphs [3].

Nevertheless, those problems are still significantly different. This work is centered around the adwords problem, or more specifically, the one-sided online maximum cardinality bipartite matching problem. The provided examples show its essential characteristics, in particular, how it differs from some other related problems. As this is a follow-up article of "On edge usage in adwords problem" [5], some terminology is provided only with intuitive meanings and is not backed with formal definitions.

The main result of [5] is Theorem 3.1 which says that in one-sided online maximum cardinality bipartite matching problem the greedy heuristic which at each step minimizes the maximum rank of an edge (the number of times any augmenting path have visited that edge since the beginning of the algorithm run) satisfies the bound of $\operatorname{rank}(e) = O(\sqrt{n})$ for any edge e. The main result of the current paper is an example which matches the bound from Theorem 3.1 of [5], hence, proves its tightness. In section 3.2 an example is shown where the maximum rank of an edge is of square root order with regard to the number of vertices. Nevertheless, we encourage to go over all the provided examples as they provide not only building-blocks, but also intuition why the result holds.

We end this introduction with brief description of the provided examples. The next section presents the family of resource allocation problems and proves a logarithmic upper bound for one of them, namely the online load balancing problem. Section 3 follows with some preliminary instances, namely paths, trees and a slightly more involved Fibonacci-themed example which all imply logarithmic lower bounds for the maximum rank of an edge, but show the basic building blocks which would later lead to the main result. Especially, the third of them introduces the concept of allegiance switching, that is, the core feature powering the polynomial examples presented in Subsection 3.2. Finally, it ends with counterexamples for path-length-related and size-related heuristics, and the whole paper concludes in Section 4.

It is worth noting, that this paper contains many diagrams. As some examples are complicated, it would be completely futile to attempt to describe them solely by equations. Therefore, the figures became an inherent part of this work, and many details are not worded in the text. A significant effort was made to keep them as simple as possible, but sometimes the underlying pattern would not be apparent until a bigger instance was drawn. For precisely this reason, the two most important examples use color, as their sizes ensured it would be impractical to keep the pictures only in shades of gray. Thus, even if not strictly necessary, a color-enabled medium would be definitely helpful.

2 Resource allocation problem

The resource allocation problem is a close counterpart to the adwords problem, and has been described here to emphasize why the polynomial lower bound for the rank-minimizing algorithm in Corollary 3.1, is so significant. We consider one particular version, namely the online load balancing problem.

Suppose we have n servers and expect to process m tasks which arrive *online* with specification on which servers each task can be run. The challenge is to minimize the maximum load of all servers.

In this setting, as in many other online problems, the best solution might be arbitrarily bad, namely if we were to constrain all m programs to run only on the first computer, the load would be m, the worst possible, independent of how big n is. Hence, as in other online problems, the common way of assessing the solution is to make the competitive-analysis, that is, compare the online algorithm to the offline solution.

Let $G = \langle U \uplus V, E \rangle$ be a bipartite graph. The vertices of U are known in advance, while the $v_t \in V$ arrive online with their adjacent edges. Each time we need to pick an edge to match v_t to some u as to minimize the maximum degree of U. Any choice made is final, i.e. there are no further changes, in particular the result is a semi-matching as we allow multiple $v \in V$ to be assigned to a single $u \in U$. There are two major settings:

- adversarial — where v_t are given by an adversary, who can decide on next v taking into account our previous decisions;
- stochastic — where v_t come from some distribution, usually uniform.

The algorithm commonly called `GREEDY` assigns the newly arrived vertex to the server with minimum load and resolves the ties arbitrarily.

Theorem 2.1. *[1] In the adversarial setting the competitive ratio of* `GREEDY` *is* $O(\log n)$.

Proof. Let $v_t \in V$ for $t = 1, \dots, n$ be the online sequence of vertices and N the optimum offline semi-matching (the degrees of vertices of U may be bigger than one). Consider a run of `GREEDY` and the respective semi-matching M. Set u_t and w_t to be vertices of U such that $(u_t v_t) \in N$ and $(w_t v_t) \in M$. Define

$$\alpha(t) = \left|\left\{v_\tau \in V \;\middle|\; (u_t v_\tau) \in N, \tau \leqslant t\right\}\right|,$$
$$\beta(t) = \left|\left\{v_\tau \in V \;\middle|\; (w_t v_\tau) \in M, \tau \leqslant t\right\}\right|.$$

Now let $A = \max_u \deg_N(u)$ be the maximum load in the optimal solution and $B = \max_w \deg_M(w)$ be its counterpart for `GREEDY`.

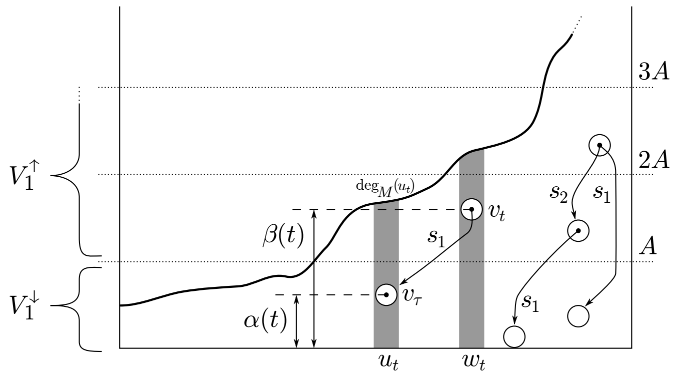
Case $A = 1$. Let

$$V_k = \{v_t \in V \mid \beta(t) = k\}.$$

Now, observe that $|V_1| \geqslant \frac{m}{2}$ and $\frac{1}{2}|V \setminus \bigcup_{i=1}^{k} V_i| \leqslant |V_{k+1}| \leqslant |V_k|$. Clearly $A = 1$ implies $m \leqslant n$, thus $B \leqslant \lceil \log_2 n \rceil + 1$ and the competitive ratio is not worse than $\frac{B}{A} = O(\log n)$.

Case $A > 1$. Set $V_k^{\uparrow} = \{v \in V \mid \beta(t) > kA\}$ and $V_k^{\downarrow} = \{v \in V \mid (k-1)A < \beta(t) \leqslant kA\}$ and consider a function $s_k(t) : V_k^{\uparrow} \to V_k^{\downarrow}$ defined as

$$s_k(v_t) = v_\tau \quad \iff \quad u_t = w_\tau \wedge \alpha(t) = \beta(\tau) - (k-1)A.$$

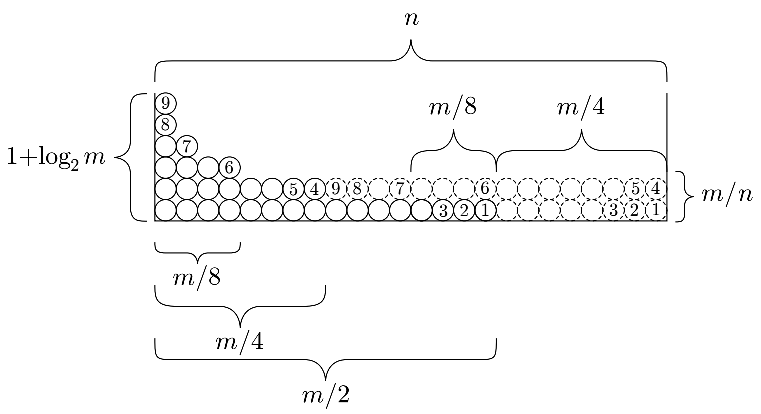


First, please note that both $\langle u_t, \alpha(t) \rangle$ and $\langle w_t, \beta(t) \rangle$ are unique. Moreover, since $\alpha(t) \leqslant A$ for any t, then s_1 represents a 1-to-1 correspondence between $V_1^{\uparrow}$ and a subset of $V_1^{\downarrow}$. Similarly, for $k > 1$ and any $v_t \in V_k^{\uparrow}$ we have that $\deg_M(u_t) \geqslant kA$, thus s_k is also bijective with its value set. Clearly, $|V_k^{\uparrow}| + |V_k^{\downarrow}| = |V_{k-1}^{\uparrow}|$, and because of s_k we arrive at $|V_k^{\uparrow}| \leqslant |V_k^{\downarrow}|$ and so $|V_k^{\uparrow}| \leqslant A\frac{n}{2^k}$. Finally, if $|V_k^{\uparrow}| < A$ then $V_{k+1}^{\uparrow} = \varnothing$ and so the sets are non-empty for $k \leqslant \lceil \log_2 n \rceil + 1$, hence $B \leqslant A\lceil 1 + \log_2 n \rceil$ and $\frac{B}{A} = O(\log n)$. $\square$

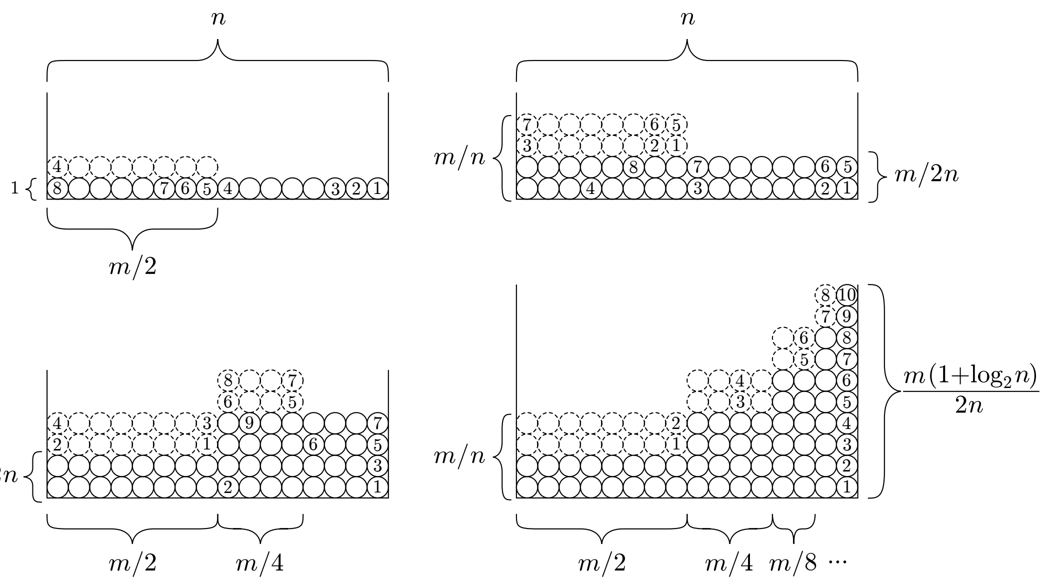
Theorem 2.2. *[1] In the adversarial setting the competitive ratio of* `GREEDY` *is $\Omega(\log n)$ for $m = \Omega(n)$.*

Proof. Without loss of generality we can assume that $n = 2^k$ for some $k \in \mathbb{N}$. The construction of a graph with $2n$ vertices that causes `GREEDY` to have load of $\Theta(k) = \Theta(\log n)$ has been presented in the following diagrams. The figures picture consecutive batches of steps of an algorithm run, each new vertex comes with two edges, some corresponding pairs were marked by the numbers. The solid circles represent `GREEDY` solution and the dotted positions are the optimal ones.

Case $m < 2n$**.** The white vertices are represented by columns and the balls are the black vertices. The diagram presents the case where $n < m < 2n$. For $m \leqslant n$ the length of layers is smaller than $\frac{n}{2}$ and as such both fit at the bottom.



Case $m \geqslant 2n$**.** Some numbers (e.g. 8 in the first three figures) were matched in an optimal manner and hence do not have a pair. Please note that the optimal solution has maximum load of $\frac{m}{n}$, hence the resulting competitive ratio is $\Theta(\log n)$.



□

Corollary 2.3. In the adversarial setting the competitive ratio of `GREEDY` is $\Theta(\log n)$.

The following theorem is beyond the scope of this paper and is cited here only for comparison. `RANKING` is an algorithm that is very similar to `GREEDY`, however it breaks the ties according to a random permutation. The proof along with many more details can be found in [1].

Theorem 2.4. *[1] For the stochastic setting using a uniform distribution, the* `RANKING` *algorithm has* $O(1 + \frac{n \log n}{m})$ *expected competitive ratio.*

3 Adwords problem

The adwords problem is a setting where we have n known advertisers and expect to process m ad-requests of some publishers which arrive sequentially *online*. Both publishers and advertisers have constraints on with whom they want to work with, i.e. the feasibility of advertisers for a request is determined once they arrive, during the algorithm run. The challenge is to maximize the number of ad impressions, i.e. semi-matchings of advertisers and requests.

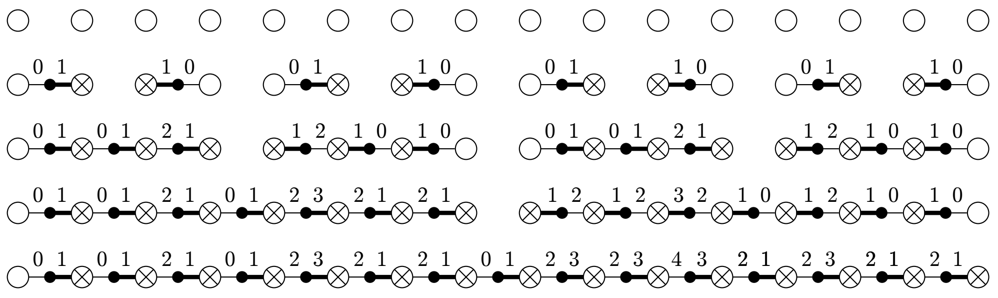
Similarly as to the online load balancing problem, the common approach is to do competitive analysis, that is, compare the performance of an algorithm to the optimal solution. However we consider a slightly different problem, namely one-sided online maximum cardinality bipartite matching. The difference is that we request the algorithm to maintain the maximum cardinality matching and for it to be possible we allow to rematch already assigned vertices. There is no point in comparing the maximum cardinality matching to the offline solution, hence we analyze the necessary number of rematchings instead. Specifically in the previous paper we have proved that the number of changes counted per edge done by the rank-minimizing algorithm is bounded by $O(\sqrt{n})$. Here we will provide examples which will show that this bound is tight.

Let $G = \langle U \uplus V, E\rangle$ be a bipartite graph. We will consider the one-sided online bipartite maximum cardinality matching problem, that is, a setting where U (i.e. its size) is known beforehand and $v \in V$ are given sequentially along with the adjacent edges during the algorithm run. The challenge is to maintain the maximum cardinality matching doing only small number of changes per edge, i.e. how many times an edge is added and removed from the matching . This number will be called the *rank* of an edge $e \in E$ at turn t and will be denoted by $\mathrm{rank}_t(e)$.

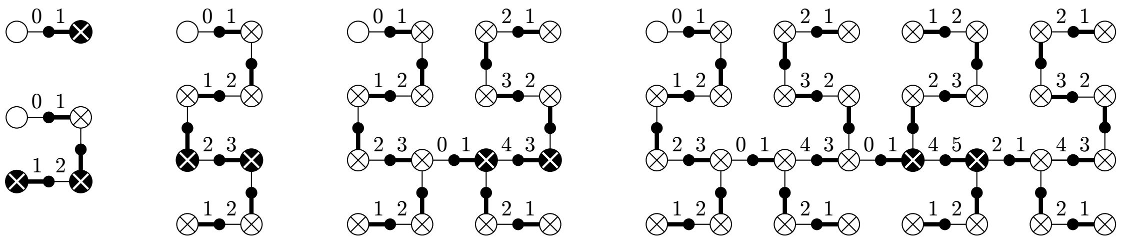
3.1 Exponential examples

This subsection considers exponential examples. There are two reasons: they are rather easy, e.g. allow to gain intuition; secondly, they introduce building block used in further sections.

Simple path. The simplest example is a path. The following diagrams picture steps 0, 8, 12, 14 and 15. The incoming vertices $v_t \in V$ are marked black, and the matched $u \in U$ are marked by a diagonal cross. The numbers near the edges represent the ranks at the given turn. Rank r is forced by an instance of size $2^{r+1} - 1$.

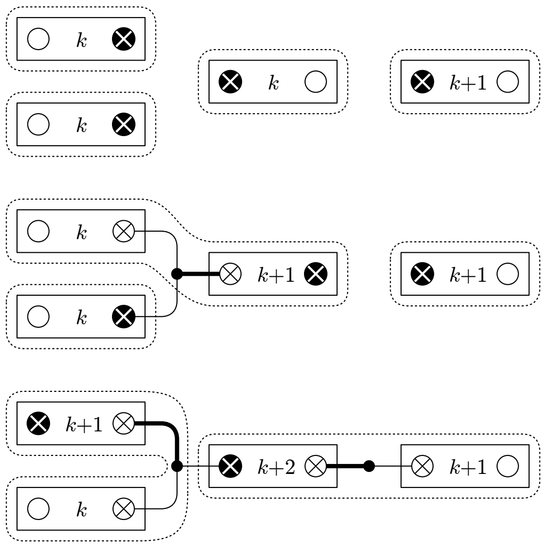


Tree of height $\Theta(r)$. The diagram shows the first five iterations of the discussed example. The vertices of the highest-ranked edge are marked by inverted colors, while the ranks of vertical edges were omitted for the sake of readability. The next iteration is constructed by connecting the matched ends of the marked edge pairs (the two adjacent to a black vertex) of two previous iteration copies (note, that in the diagram the second copy is reflected).



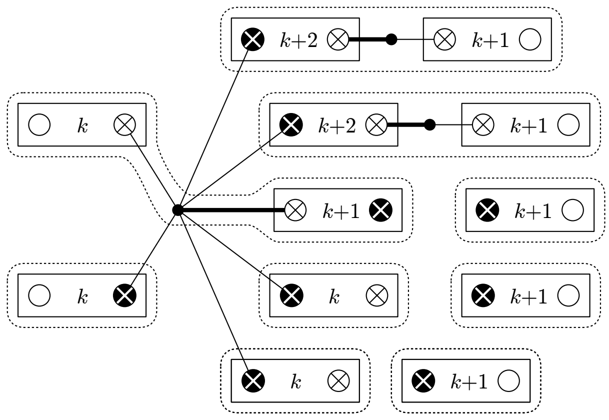
Observe, that the eccentricity of the marked vertices is linear with regard to the maximum rank (i.e. rank of the marked edge): each time we connect current two copies, it increases at most by 3×2. Therefore, the tree height is bounded by $12r$. Similarly to the path it has 2^r white vertices.

Fibonacci example. The two above examples are significant, because there is always only one simple augmenting path, in particular implied lower bounds are valid for *any* augmenting path algorithm. However, if we drop this assumption, the construction can be improved so it approaches ϕ^r, where ϕ is the golden ratio, i.e. $\phi = \frac{1+\sqrt{5}}{2}$. The steps of obtaining the crucial feature are as follows:



The dotted lines denote borders of areas for possible augmenting paths, intuitively the "basins" of unmatched vertices. The inverted-color vertices are those which would cause an increase of rank in the dotted subgraphs, if matched. We start with $(n + 2)$ components of rank k and n components of rank $k + 1$, after n connections we obtain a component of rank $k + 1$ and n components of rank $k + 2$,

$$(n+2)\cdot C_k + n\cdot C_{k+1} \rightsquigarrow 1\cdot C_{k+1} + n\cdot C_{k+2}.$$



Observe, that with n going to infinity this scheme approaches $C_k + C_{k+1} \rightsquigarrow C_{k+2}$. Curiously, it seems that the augmenting path joining C_k and C_{k+1} goes "the wrong way" and increases the rank of the larger component. That particular behavior is explained by the fact that in the second step there are two components of which rank increases (naturally $k+1 \rightsquigarrow k+2$, but also $k \rightsquigarrow k+1$). Even if at first the smaller one was in the same dotted area as the bigger component, it switches its allegiances. In other words, a single augmenting path may potentially cause many subgraphs to increase their ranks, the reason being their boundaries changes.

The introduced rule is easily transformed into a concrete example by following it backwards.

$$\begin{aligned}
1\cdot C_r &\leftsquigarrow 2\cdot C_{r-1} && \Big(+1\cdot C_{r-2}\Big)\\
&\leftsquigarrow 4\cdot C_{r-3} + 2\cdot C_{r-2} && \Big(+1\cdot C_{r-3}\Big)\\
&\leftsquigarrow (2+2)\cdot C_{r-4} + (4+2)\cdot C_{r-3} && \Big(+1\cdot C_{r-4}\Big)\\
&\leftsquigarrow (6+2)\cdot C_{r-5} + (4+6)\cdot C_{r-4} && \Big(+\dots\Big)\\
&\vdots\\
&\leftsquigarrow \alpha\cdot C_k + \beta\cdot C_{k+1} && \Big(+1\cdot C_k\Big)\\
&\leftsquigarrow (\beta+2)\cdot C_{k-1} + (\alpha+\beta)\cdot C_k\\
&\vdots\\
&\leftsquigarrow (4F_r)\cdot C_1 + (4F_{r+1}-2)\cdot C_2 && \text{where } F_k \text{ is the } k\text{-th Fibonacci number}
\end{aligned}$$

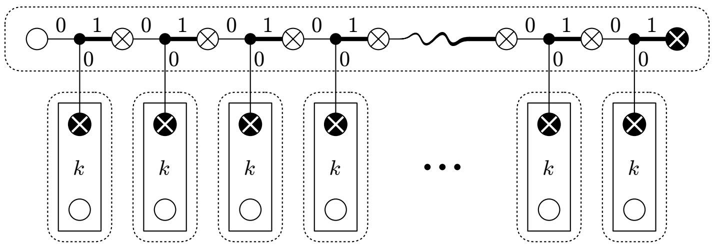
It is worth noting, that in this example, each vertex has at most one path that leads to an unmatched counterpart and minimizes the maximum of ranks of travelled edges. Hence, no algorithm that follows this heuristic can obtain better bounds than $r = \Omega(\log_\phi n)$. The most important feature of this example is that one augmenting path increases ranks of multiple subgraphs. It is precisely this behavior that we will exploit to obtain examples of polynomial size.

3.2 Polynomial examples

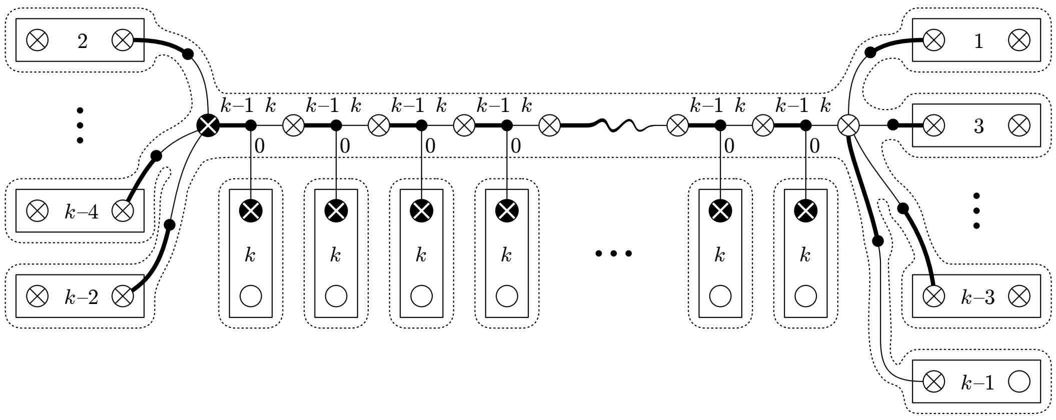
Here we will construct two polynomial examples, one of size $\Theta(r^3)$ and the second of size $\Theta(r^2)$.

First of them is presented as an intermediate step, however, it is noteworthy because it has some sort of robustness, e.g. one of its features is that for each vertex there is at most one turn for which a dubious choice happens (i.e. the algorithm has to pick between an optimal and non-optimal solution).

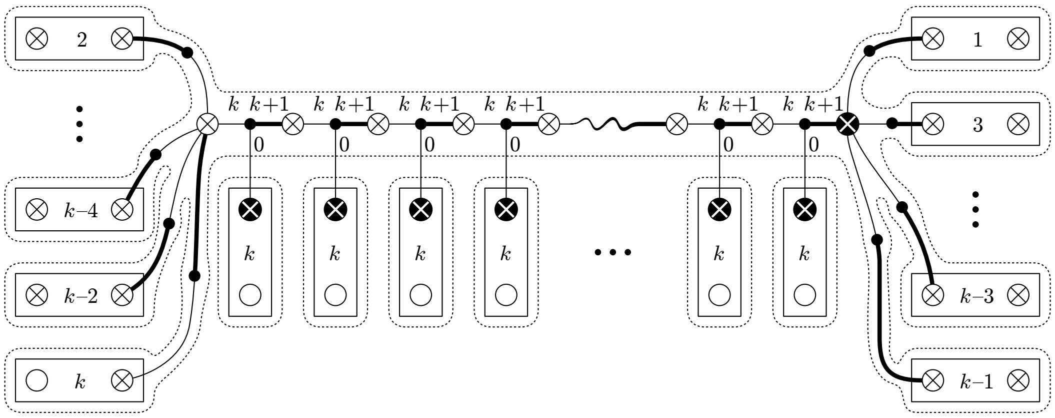
The comb. The basic building block of the first example is a structure we will call a "comb". The k-ranked comb of length n consists of a long low-ranked path joining n components of rank k.



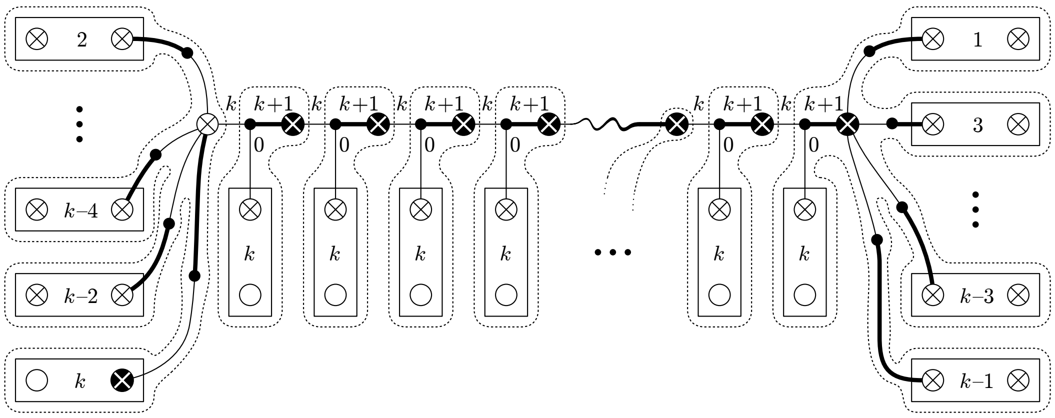
Then we will "fatten" the comb by supplying components of rank $1, 2, \ldots, k-1$ on its both sides.



Finally, we will add the most important piece, the component of rank k. This is the moment, when the algorithm performs a suboptimal choice. It can pick any path of rank k, that is, it is possible that in the worst case it will pick the longest one.

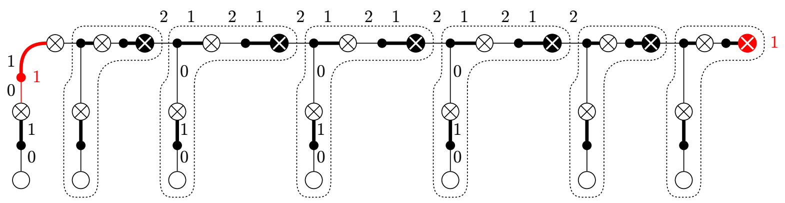


However, after this step, the edges of rank $(k+1)$ can choose a better connection, i.e. they switch their allegiances to the n components we have prepared at the beginning. It is this characteristic (a single augmenting path that causes multiple components to increase their ranks) that makes the polynomial example possible.

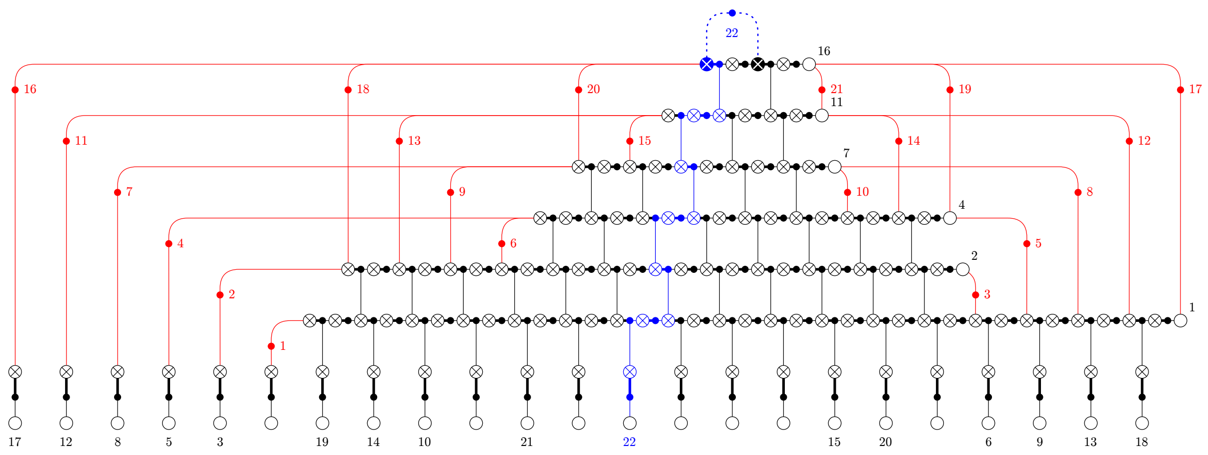


We have used $n+1$ components of rank k and one component for ranks $1, 2, \ldots, k-1$; on the other hand we have created n components of rank $k+1$, available for further use.

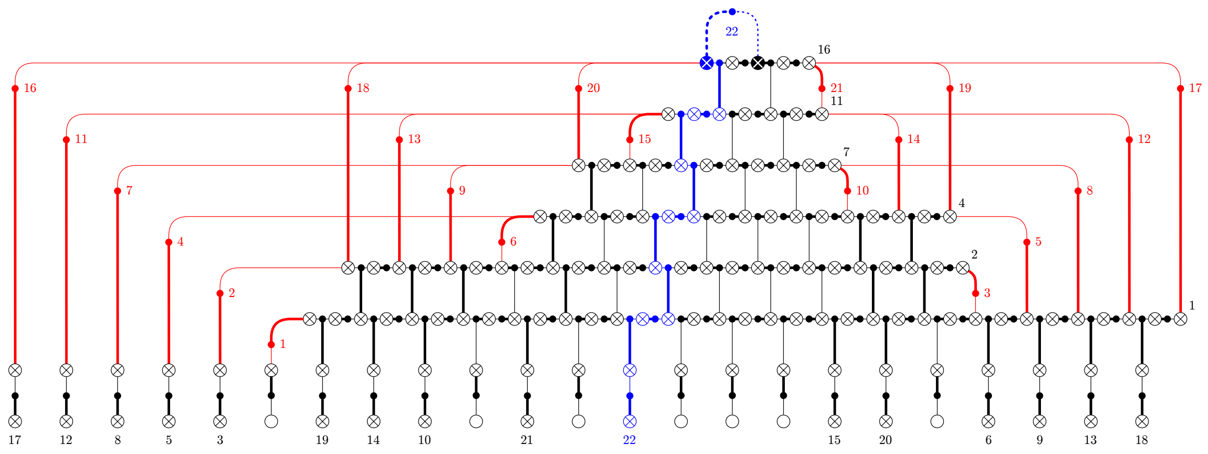
A concrete example of a comb may look as below, it consists of $6+1$ components of rank 1 and provides 6 components of rank 2. For technical reasons we add some intermediate vertices so that any augmenting path starting in any of the marked vertices would have to go through an edge of rank 2 (observe, that if it were to start at the node immediately left of the marked one, then the biggest rank of the shortest path to an unmatched vertex is only 1).



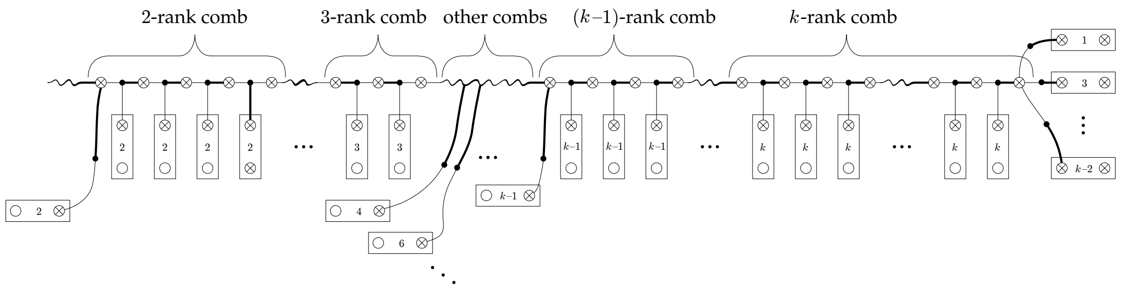
The $\Theta(r^3)$ example. This example is generated by stacking r combs on top of each other, i.e. using the new components of rank $k+1$ for the next comb of that is shorter by one element. Note, that we need to provide not only the comb-components, but also the side-ones that make the comb fat. This causes the example to have size of order $\theta(r^3)$, optimizing it will lead to better bounds, but would also immensely complicate the graph.



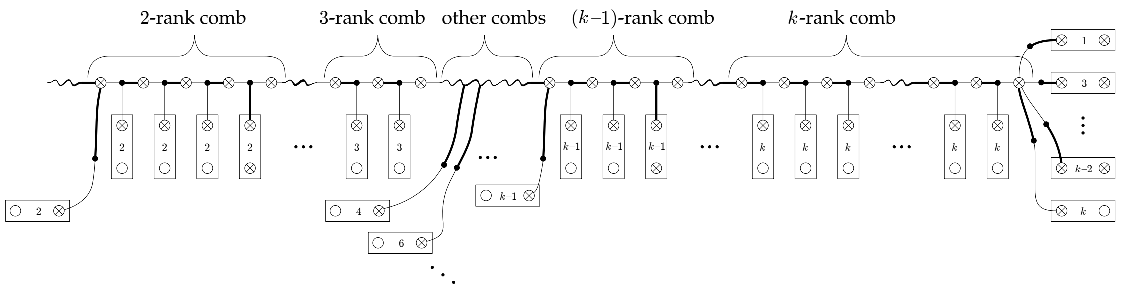
The first of two diagrams pictures the state before applying side-components. The numbers indicate corresponding pairs of black and white vertices, that is, those that were matched in given turn. The endpoints of a path containing the highest-ranked edges were marked by inverted colors. Although there is a number of unmatched vertices, the size is still of order $\Theta(r^3)$.



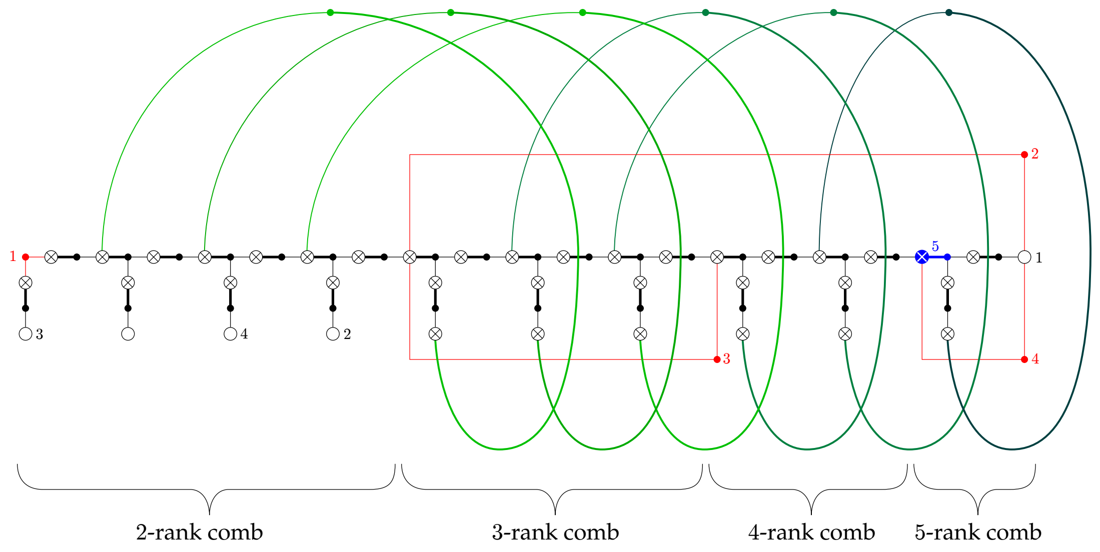
The $\Theta(r^2)$ example. The issue with the previous instance is that we require multiple components for each of the stacked combs to make them fat. However, an inquisitive reader may observe that we are doing the same job many times, that is, each comb of order bigger than k has to be pushed from rank k to $k+1$. Indeed, we could improve the example so that a single component of rank k would be enough to make all the appropriate combs into $k+1$. To achieve it, we will join all the combs in a sequential manner so that a single, long augmenting path could go through them all, as in the sketch below. This modification will be the base of our compression and will allow us to construct an example of size $\Theta(r^2)$.



To obtain structure similar to the one in previous paragraph, we apply the components only to appropriate combs by joining them in the middle of the chain. In the diagrams k is odd, which means that after the k-th pass (the diagram below) the path in the graph will lead from left to right (similarly for 3-rank comb), while the part of paths near combs of even rank goes from right to left, as for 2-rank and $(k-1)$-rank combs.

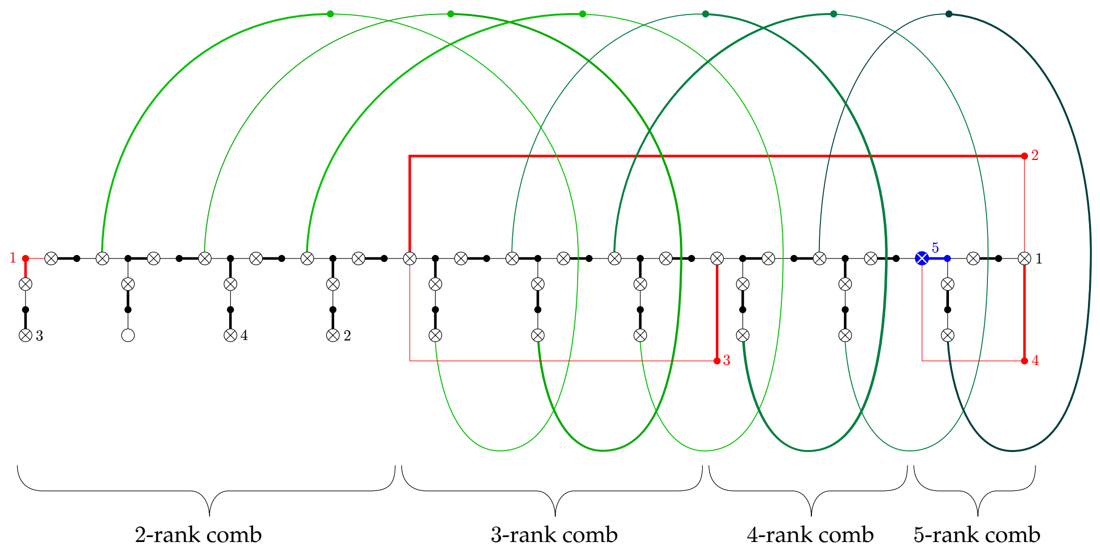


The whole concrete example might look as follows, here for $r = 5$. The green edges are those which would stack two combs on top of each other in the $\Theta(r^3)$ case. Observe, that 4-rank comb does not need additional backing by 2-rank components, as the 3-ranked provide it while it is necessary, i.e. at that time those 3-ranked components are of rank 2. The red edges are responsible for making the combs fat, and the blue one is the final highest-ranked edge.



The four augmenting paths go, as noted in corresponding points. The final state can be seen in the next figure.

1. From vertex marked by red number 1, straight right, by the black path, until its black counterpart.
2. Starting near red 2, down onto the black path, and then left until junction leading to black 2, where it finishes.
3. From red 3 vertex, one step up, then right until the end, via red 2, back onto black path, one step right, and through green arc, finally reaching vertex labeled by black number 3.
4. Up to black 1 and left, through the blue vertex, then further left, until the first junction down which would lead by two green arcs and small number of black edges to the black pair of red 4.



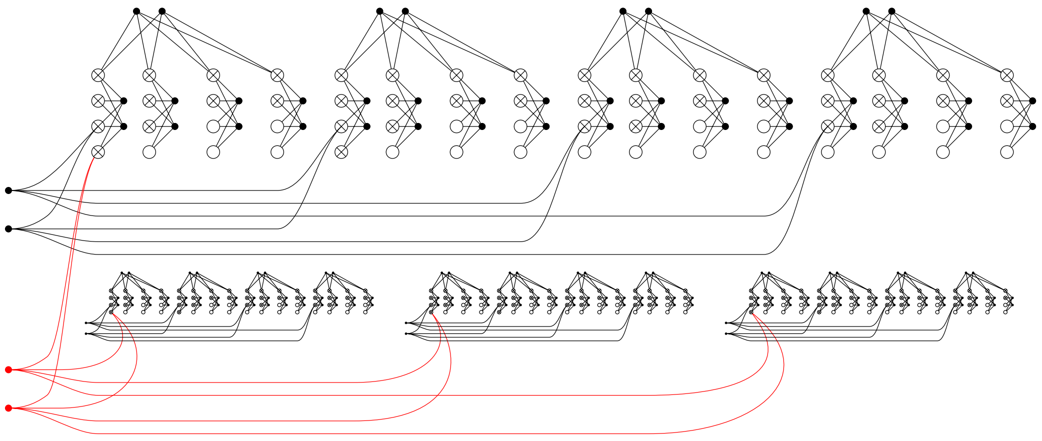
The boundaries between combs are blurry, because of frequent orientation changes of edges. However, we can still count the number of white vertices, and so, the 5-rank comb uses 4, 4-rank comb uses $2 \cdot 4$, the next one $3 \cdot 4$ and the last, 2-rank comb uses $4 \cdot 4$. Generalizing for arbitrary r, we have that k-rank comb needs $(r-k+1) \cdot 4$ nodes and the sum $4 \sum_{k=2}^{r} r-k+1 = 4 \sum_{k=1}^{r-1} k = 2r(r-1)$ makes the quadratic nature of the considered example self-evident.

Corollary 3.1. The bound in Theorem 3.1 of [5] is tight.

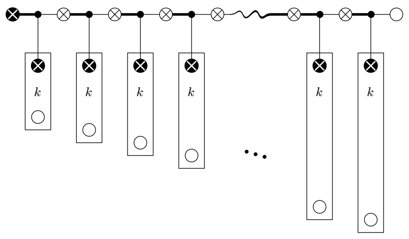
3.3 Miscellaneous examples

In this section we will briefly discuss some additional examples not directly related to the problem, but nevertheless relevant. Some will argue that heuristics are not as promising as it seems, others will emphasize the peculiar nature of the problem.

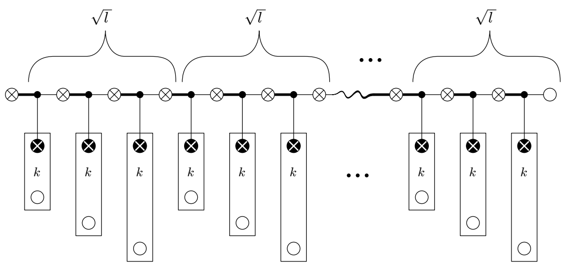
Exponential with loose unmatched vertices. The example below illustrates that it is quite easy to construct a graph which would require arbitrarily high ranks, even with some additional assumptions, like some loose unmatched vertices for each black vertex, or bounds on numbers ratio of white vertices to black vertices, or even some structural properties (here each black vertex has an isomorphic pair).



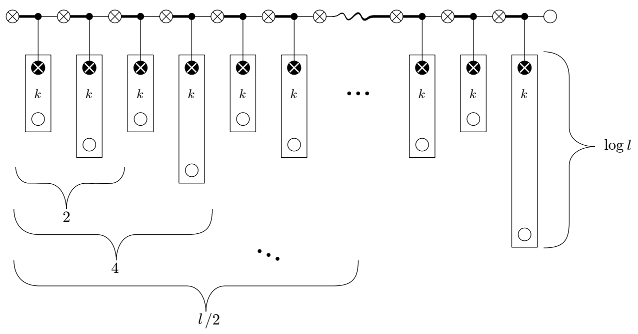
On the counterexamples for path-length-related heuristics. It may seem that some of the counterexamples, in particular the $\Theta(r^2)$ example, could be refuted by adding some heuristics related to path-length, e.g. to prefer shortest paths or paths with smallest number of high-ranked edges, and so on. The natural way to adapt our examples to fool a heuristic choosing a shorter path over longer one is to increase the lengths of the teeth of the comb along the augmenting path as shown in the figure below.



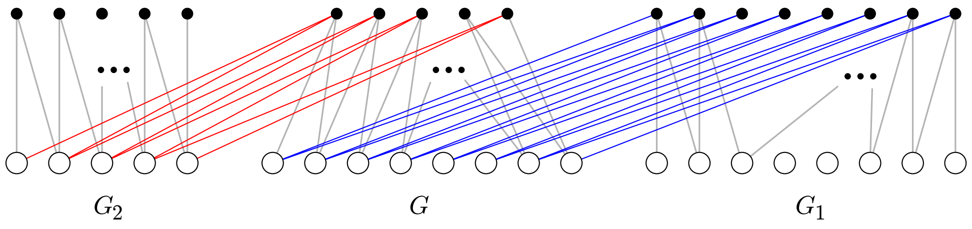
If applied to the $\Theta(r^2)$ example, this method blows up the number of vertices to $\Omega(r^3)$. To create an instance of order smaller than $O(r^3)$, one could consider the following approach. Each comb is split into parts of size of square-root of the length of the comb. This allows for multiple side-components of the same rank to be applied in the middle of a comb, hence minimize the necessary number of vertices.



For some heuristics, it might be better to apply a similar techniques based on exponential structures like the one in the next diagram.



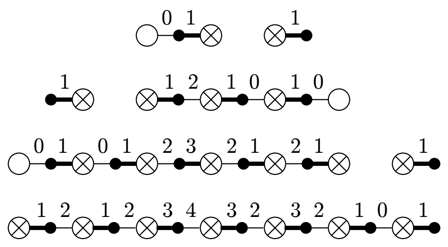
On the counterexamples for degree-related heuristics. The heuristics dealing with vertex degree do not work as well, the primary reason being, we can join arbitrary graph with some external features that would not change the matching-properties, but would totally disrupt the degrees of the vertices. One way of doing that is as follows. Let G be a bipartite graph of $n_1 + n_2$ vertices, then add two new arbitrary bipartite graphs G_1 and G_2 on $n_1 + n_1$ and $n_2 + n_2$ vertices respectively, both of which contain a perfect matching. Then connect all black vertices of G_1 to white vertices of G and all white vertices of G_2 to black in G in a way that would disturb the degrees of G the most. Naturally, the matching of G_1 still has to be contained in G_1 and similarly for G_2, therefore the maximum cardinality matching of the newly constructed graph is precisely $n_1 + n_2$ bigger than the original one.



Random subgraph of a graph with triangular adjacency matrix. Examples provided in this work regard mostly the worst-case analysis. Although expected time complexity is beyond the scope of this paper, we can note that experimental tests show that an implementation of rank-minimizing algorithm in practice performs significantly better that the worst-case bound would suggest.

As for now, the hardest cases in the sense of sampled expected time are the random subgraphs of graphs with triangular bipartite adjacency matrix and their slight modifications. Exact expected time complexity is not yet known and requires further investigation.

An example for linear rank in full online maximum cardinality bipartite matching. This example argues that the full (or two-sided) online maximum cardinality bipartite matching is substantially different than its one-sided version and the adwords problem. It manifests in the fact that it is possible to construct an example that would require ranks of linear order to maintain the maximum cardinality matching. We start with a simple path, and make it alternate left and right, the first few steps are shown below. Please note, that we don't need to assume that the ties are resolved arbitrarily, in fact there is unique augmenting path each turn.



The essential distinction is that in adwords problem all the white vertices are available from the start, hence, if there is no available augmenting path that would use edges of rank k or smaller, there won't be any anymore. In particular, if there is no augmenting path at all (the vertex would have to be left unmatched), then it will stay that way until the end.

On the other hand, here we can provide additional unmatched vertices when necessary, and so fool the algorithm to go the costly way, and then again back to the newly available vertex. Iterating this tactics would lead to arbitrarily high ranks using only constant number of edges and vertices per step.

4 Conclusion

This paper presents several examples for the one-sided online maximum cardinality bipartite matching problem. Case by case we show the characteristics of rank-minimizing algorithms and introduce insights about hard cases in the considered task. This provides the intuition behind the proof of Theorem 3.1 of [5], significantly, one of the examples matches the bound and so proves it tight.

However, all the presented examples deal with the worst-case analysis and depend on particular order of edges and vertices. Indeed, experiments show that the rank-minimizing algorithm performs in practice significantly better, especially if the input is in random order. Clearly, the question whether its expected time complexity is of strictly smaller order or perhaps it is just a mere coincidence, requires further investigation.

References

[1] Y. AZAR, *On-line load balancing*, in Online Algorithms, A. Fiat and G. J. Woeginger, eds., vol. 1442 of Lecture Notes in Computer Science, Springer, 1996, pp. 178–195.

[2] J. EDMONDS, *Paths, trees and flowers*, Canadian Journal of Mathematics, 17 (1965), pp. 449–467.

[3] J. E. HOPCROFT AND R. M. KARP, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM Journal on Computing, 2 (1973), pp. 225–231.

[4] H. W. KUHN, *The hungarian method for the assignment problem*, Naval Research Logistics Quarterly, 2 (1955), pp. 83–97.

[5] D. LENIOWSKI, *On edge usage in adwords problem*, 2013. The second progress report for SSDNM, available at `http://ssdnm.mimuw.edu.pl/`.