Chapter 1

Port Numbering Model

In this chapter, we will look into a fundamental question in distributed computing that was asked in a paper by Dana Angluin in 1980: "How much does each processor in a network of processors need to know about its own identity, the identities of other processors, and the underlying connection network in order for the network to be able to carry out useful functions?"[Ang80]. In her paper, Dana Angluin introduces the port numbering model. This model makes few assumptions about the underlying network: processors are arranged in a graph and can communicate with their peers by exchanging messages pairwise. There is no global network setup and thus the processors cannot identify themselves to others. We start with negative news showing that some very basic graph problems cannot be solved in such a model. On the positive side, there are also exciting graph problems that can be solved in this model efficiently.

1.1 Graph problems

Definition 1.1 (Matching)**.** *Given an undirected graph $G = (V, E)$, select a set of edges M such that no two selected edges are connected to the same node, that is, $e_1 \cap e_2 = \varnothing$ for all $e_1, e_2 \in M$.*

Definition 1.2 (Vertex cover)**.** *Given an undirected graph $G = (V, E)$, select a subset of nodes $X \subseteq V$, such that each edge has at least one vertex in X, i.e., $e \cap X \neq \varnothing$ for all $e \in E$.*

Definition 1.3 (Vertex coloring)**.** *Given an undirected graph $G = (V, E)$, assign a color c_v to each vertex $v \in V$ such that the following holds: $e = \{v, w\} \in E \Rightarrow c_v \neq c_w$.*

Remarks:

- Throughout this course, we use the terms *vertex* and *node* interchangeably.
- Observe that the presented problems admit trivial solutions: add no edges to the matching, add all nodes to the vertex cover, or color each node of the graph with its own color.

- In this chapter, we will be interested in the optimization versions of the above problems: What is the maximum number of edges that can be added to a matching? (*maximum matching problem*); What is the smallest number of nodes that are needed for a vertex cover? (*minimum vertex cover problem*); Can we color a given graph with k colors? (k*-coloring problem*)
- Vertex coloring is an important graph problem that is used to model different scheduling problems. In the distributed setting, vertex coloring can help us organize the execution of tasks thus making sure that neighboring nodes do not execute a task at the same time.
- In distributed computing, we often take the point of view that the system is a graph whose nodes are processors and whose edges both represent relations with respect to the problem at hand and communication links. This will come in handy here as an abstraction, but in many systems, it is literally true.

1.2 Model

Definition 1.4 (Degree)**.** *The neighbors of a node v are denoted as $N(v)$. Node v has $\delta(v) = |N(v)|$ neighbors, which is called the* degree *of v. The maximum degree vertex in a graph G defines the graph degree $\Delta(G) = \Delta$.*

Definition 1.5. *A* port-numbered network $G = (V, E, \{p_v\}_{v \in V})$ *is described by a graph (V, E) where for each node $v \in V$ there is a bijection $p_v : \{w \in V | \{v, w\} \in E\} \to \{1, \ldots, \delta(v)\}$.*

Remarks:

- Figures 1.6 and 1.7 visualize different representations of the same port-numbered network.

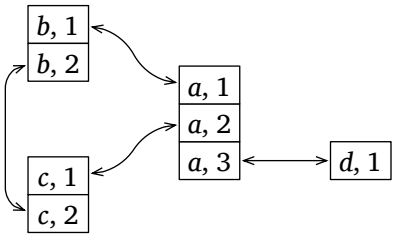

Figure 1.6: A port-numbered network $G = (V, E, \{p_v\}_{v \in V})$. There are four nodes, $V = \{a, b, c, d\}$; the degree of node a is 3, the degrees of nodes b and c are 2, and the degree of node d is 1. The connection function p is illustrated with arrows—for example, $p_a(d) = 3$ and conversely $p_d(a) = 1$. This network is simple.

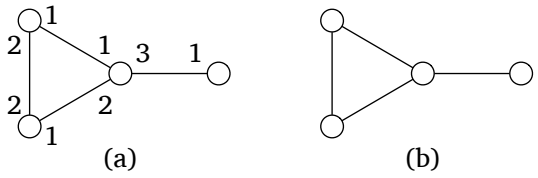

Figure 1.7: (a) An alternative drawing of the simple port-numbered network G from Figure 1.6. (b) The underlying graph without a port numbering.

Definition 1.8 (Synchronous distributed algorithm)**.** *In a synchronous distributed algorithm, nodes operate in synchronous rounds. In each round, each node executes the following steps:*

1. *Send messages (of reasonable size) to each port i.*
2. *Receive messages from each port (that were sent by neighbors in step 1 of the same round).*
3. *Do some local computation (of reasonable complexity).*

Remarks:

- What does "reasonable" mean in this context? We are somewhat flexible here, and different model variants exist. Generally, we will deal with algorithms that only do very simple computations (a comparison, an addition, etc.). Exponential-time computation is usually considered cheating in this context. Similarly, sending a message with a node ID, or a value is considered okay, whereas sending really long messages is fishy. We will have more exact definitions later when we need them.
- For now, we will only look at deterministic distributed algorithms. In such algorithms, step 3 of a synchronous round can be viewed as the (same) deterministic state machine that each node executes locally.

Example 1.9 (Matching with two nodes)**.** *Assume we have two nodes in the network, connected by an edge. In the port-numbered network, each node has a single edge with port number* 1 *that is connected to the port* 1 *of the other node. To solve matching in this network, each node can send a message suggesting to join the matching via port* 1*. Once it receives the same message from the neighboring node, it can match the only edge as the matching edge.*

Theorem 1.10. *In some cases, it is impossible to solve*

- *proper vertex coloring,*
- *minimum vertex cover, and*
- *non-empty matching*

in the port numbering model using a deterministic synchronous distributed algorithm. This is because we cannot do symmetry breaking in this model.

Proof. To show that proper coloring is not possible, consider the same graph as in Example 1.9. In this graph, the nodes will execute the same steps in every round and will arrive at the same new state at the end of each round.

While vertex cover can be computed in the same graph, the minimum vertex cover only contains one node. If we could assign one node to the vertex cover in this graph, we would also solve vertex coloring, which is not possible.

For matching, we need to consider a different graph: assume that we have three nodes u, v, w that are connected to each other, where $p_u(v) = p_v(w) = p_w(u) = 1$ and $p_u(w) = p_w(v) = p_v(u) = 2$. Now the situation for each node is symmetric: if each node suggests adding its port 1 edge to the matching, they will receive the same suggestion from their port 2 edge. That is, the nodes can either add all (not a proper solution) or no edges to the matching. □

Remarks:

- For the same reasons, many other distributed problems are impossible to solve in the port-numbering network, such as maximal independent set, edge coloring, dominating set, or leader election.
- Theorem 1.10 can even be extended to graphs where all nodes have the same number of neighbors d.
- The proof presented here is not precise, we will formalize tools to prove such negative results later in the course.
- In order to break symmetries, we will investigate various options in this course, such as assigning unique identifiers to nodes, using randomness, relaxing the problem statements, or restricting the graph classes. In the following, we will focus on the special graph class: bipartite graphs.

1.3 Bipartite Maximal Matching

Definition 1.11 (Bipartite graph)**.** *A* bipartite graph *is a graph that can be colored with two colors.*

Remarks:

- In this chapter, we will call *black* and *white* the two subsets that divide the nodes of a bipartite graph. Hence, by definition, there should not exist an edge between two white (resp. black) nodes.
- Bipartite graphs appear in many applications: for example in relations such as client/server, VIP/fan, or hypergraphs,[1] where we represent each hyperedge by a node (on one side of the bipartite graph) connected to its constituent nodes (on the other side).

Definition 1.12 (Maximal matching)**.** *Given an undirected graph $G = (V, E)$ and a set of matching edges M, we say that a matching M is* maximal *if it is not possible to add more edges to M without violating the matching constraints.*

[1] A hypergraph is a structure $H = (V, E)$ in which each *hyperedge* $e \in E$ is an arbitrary subset of the nodes.

Algorithm 1.13 Matching in 2-colored graphs of maximum degree Δ using port numberings. Nodes return their matched port or $\perp$ if none of their incident edges is in the matching.

```
1: White nodes u: matching port m = ⊥
2: Black nodes v: X(v) = 1, ..., δ(v), M(v) = ∅
3: for round r = 1, 2, ... do

   Each white node u executes the following code

4:   if r = 2k − 1 then
5:     if m = ⊥ and k ⩽ δ(u) then
6:       Send propose to port k
7:     else if m ≠ ⊥ then
8:       Send matched to all ports
9:       return m
10:    else
11:      return ⊥
12:    end if
13:  end if

14:  if r = 2k then
15:    Receive incoming messages
16:    if Received accept from port i then
17:      Set matching port m = i
18:    end if
19:  end if

   Each black node v executes the following code

20:  if r = 2k − 1 then
21:    Receive incoming messages
22:    if Received matched from port i then
23:      remove i from X(v)
24:    end if
25:    if Received propose from port i then
26:      add i to M(v)
27:    end if
28:  end if

29:  if r = 2k then
30:    if M(v) ≠ ∅ then
31:      Let i = min M(v)
32:      Send accept to port i
33:      return i
34:    end if
35:    if X(v) = ∅ then
36:      return ⊥
37:    end if
38:  end if
39: end for
```

Remarks:

- Note that a maximal matching is not unique, and different maximal matchings can have different sizes.
- Algorithm 1.13 proposes a solution for computing a maximal matching in the port numbering model.
- Figure 1.14 shows a sample execution of the algorithm on a small port-numbered network.

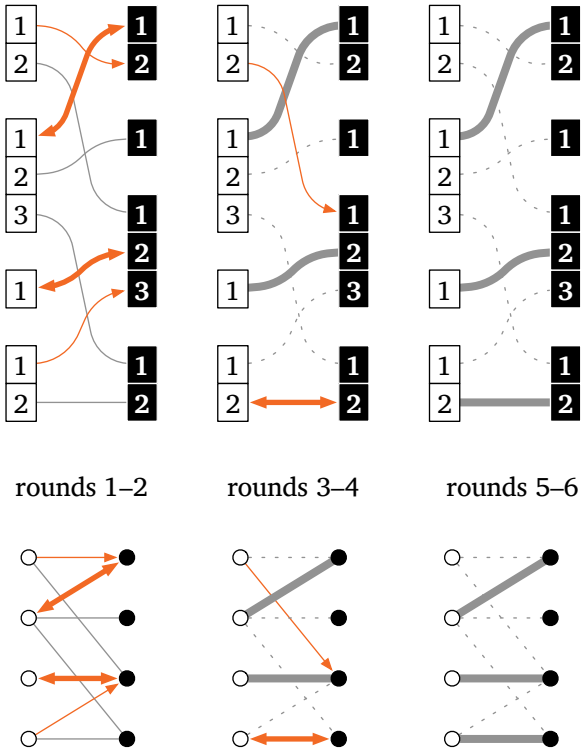

Figure 1.14: The bipartite maximal matching algorithm; the illustration shows the algorithm both from the perspective of the port-numbered network N and from the perspective of the underlying graph G. Arrows pointing right are proposals, and arrows pointing left are acceptances. Wide gray edges have been added to matching M.

Lemma 1.15. *Consider Algorithm 1.13. Assume that u is a white node, v is a black node, $p_u(v) = i$, and $p_v(u) = j$. Then at least one of the following holds:*

1. *element j is removed from $X(v)$ before round $2i$,*
2. *at least one element is added to $M(v)$ before round $2i$.*

Proof. Assume that we still have $M(v) = \varnothing$ and $j \in X(v)$ after round $2i - 2$. This implies that v is still not matched, and u has not sent *matched* to v. In particular, u is either unmatched, or matched over port i after round $2i - 2$. In

the former case, u sends *propose* to v on round $2i-1$, and j is added to $M(v)$ on round $2i-1$. In the latter case, u sends *matched* to v on round $2i-1$, and j is removed from $X(v)$ on round $2i-1$. □

Lemma 1.16. *Algorithm 1.13 stops in time $2\Delta+1$, where Δ is the maximum degree of G.*

Proof. A white node of degree d stops before or during round $2d+1 \leqslant 2\Delta+1$.

Now let us consider a black node v. Assume that we still have $j \in X(v)$ on round 2Δ. Let $p_u(v) = i$, and $p_v(u) = j$; note that $i \leqslant \Delta$. By Lemma 1.15, at least one element has been added to $M(v)$ before round 2Δ. In particular, v stops before or during round 2Δ. □

Lemma 1.17. *Algorithm 1.13 finds a maximal matching in any bipartite graph.*

Proof. Let us first verify that the output correctly encodes a matching. In particular, assume that u is a white node, v is a black node, $p_u(v) = i$, and $p_v(u) = j$. We have to prove that u matches over port i if and only if v matches over port j. If u chooses i to be its matching port, it has received an *accept* message from v, and thus v terminated with a matching over port j. Conversely, if v chooses j to be its matching port, it has received a *propose* message from u and it sends an *accept* message to u, after which u terminates with a matching over port i.

Let us then verify that M is indeed maximal. If this was not the case, there would be an unmatched white node u that is connected to an unmatched black node v. However, Lemma 1.15 implies that at least one of them becomes matched before or during round 2Δ. □

Remarks:

- Algorithm 1.13 is not very fast if Δ is large. However, if Δ is a constant, so is the running time of the algorithm.
- The assumption that we have a 2-coloring does some initial symmetry breaking for us. For instance, we already have the best coloring one could get.

1.4 3-Approximating Minimum Vertex Cover

In this section, we will relax the problem statement of the minimum vertex cover problem in order to find a solution for this problem on any graph.

Definition 1.18 (α-approximation)**.** *Let $\alpha \geqslant 1$, we say that an algorithm is a* α-approximation *for a given problem if that algorithm returns a solution that is always at most α times worse than the optimal solution.*

Remarks:

- There is a simple way to achieve a 2-approximation of minimum vertex cover: compute a maximal matching and add all the endpoints of the matching edges to the vertex cover. Matching edges do not have common vertices, and any vertex cover has to cover all edges, in particular all matching edges. However, we have not learned any distributed algorithm so far that can compute a maximal matching on any input graph.
- The algorithm that we will see in this section will only provide a 3-approximation of the minimum vertex cover, but it makes no assumption on the input. In order to be able to use Algorithm 1.13 we will create a larger, virtual bipartite graph.

Definition 1.19 (bipartite duplication)**.** *Let $G = (V, E, \{p_v\}_{v \in V})$ be a port-numbered network. A* bipartite duplication *of G is a port-numbered network $G' = (V', E', \{p'_v\}_{v \in V'})$ constructed as follows.*

1. *we double the number of nodes—for each node $v \in V$ we have two nodes v_1 and v_2 in V':*

$$V' = \{\, v_1, v_2 : v \in V \,\},$$
$$E' = \big\{\{v_i, w_j\} : \{v, w\} \in E, i \neq j\big\}$$

2. *Then we define the port numbers. If $p_u(v) = i$ and $p_v(u) = j$, we set*

$$p_{u_1}(v_2) = i \quad \text{and} \quad p_{v_2}(u_1) = j$$
$$p_{u_2}(v_1) = i \quad \text{and} \quad p_{v_1}(u_2) = j.$$

Remarks:

- With these definitions we have constructed a network G' that is bipartite by construction. We can color G' with two colors 1 and 2 as follows:
$$c(v_1) = 1 \text{ and } c(v_2) = 2 \text{ for each } v \in V.$$
Nodes of color 1 are called *white* and nodes of color 2 are called *black*.
- Figure 1.20 shows an example of a bipartite duplication and the corresponding 2-coloring.
- Note that G' also has the same degree as G.
- We can use the physical network G to efficiently *simulate* the execution of another distributed algorithm on G' without overhead in runtime.

Algorithm 1.21 Algorithm computing a 3-approximation of the minimum vertex cover.

1: Construct the bipartite duplication of the network, call it N'.
2: Simulate Algorithm 1.13 in N'. Each node v waits until both of its copies, v_1 and v_2, have stopped.
3: Node v says it is part of the vertex cover if at least one of its copies v_1 or v_2 becomes matched.

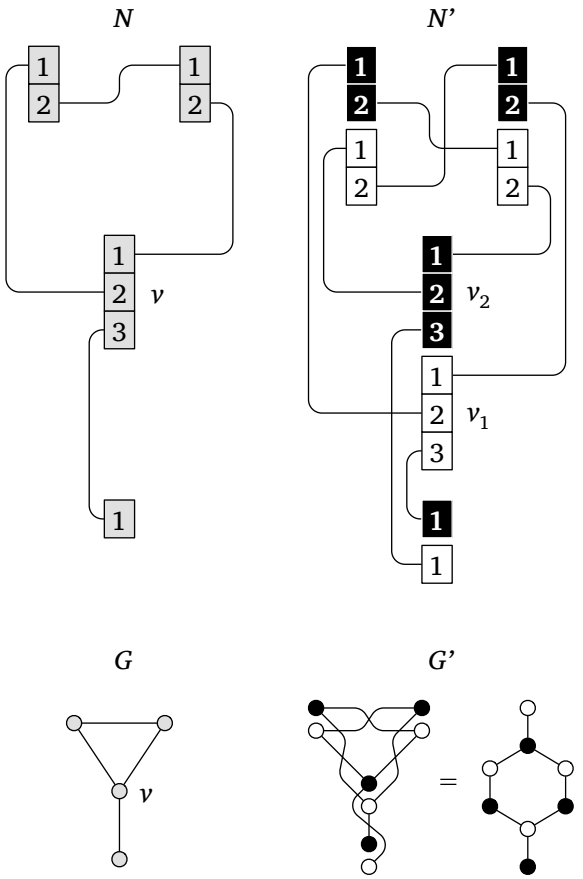

Figure 1.20: Construction of the virtual network G' in the minimum vertex cover 3-approximation algorithm.

Remarks:

- We can run this minimum vertex cover 3-approximation algorithm in any port-numbered network, as we do not make any assumptions on additional input.
- Clearly, Algorithm 1.21 stops as the bipartite maximal matching algorithm stops. Moreover, the running time is $2\Delta + 1$ rounds, where Δ is the maximum degree of G.
- Figure 1.22 shows how such a computed vertex cover looks for the network from Figure 1.20.

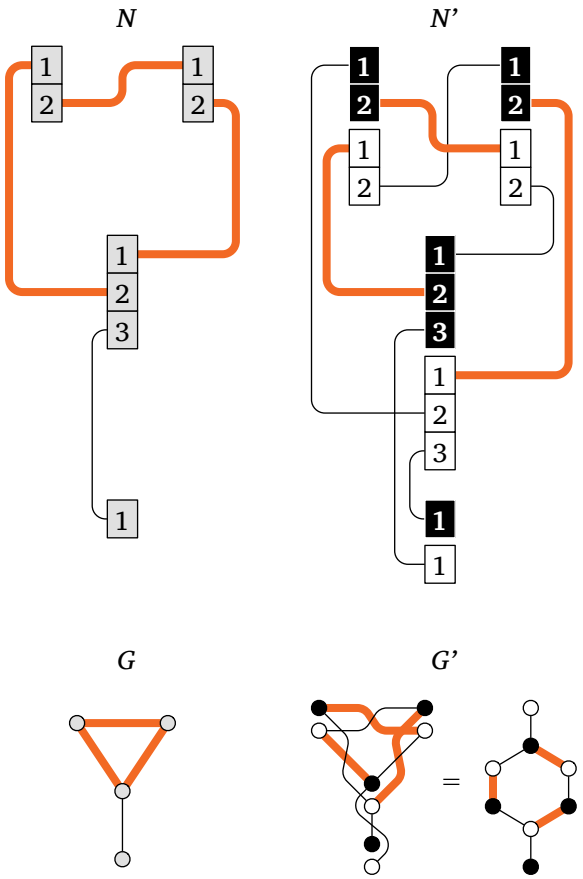

Figure 1.22: The computed matching in G' and the corresponding edges in G.

Theorem 1.23. *Algorithm 1.21 computes a vertex cover of a graph G.*

Proof. To see this, first remember that our matching algorithm produces a maximal matching, which means it is not possible to select another edge in G' without having two selected edges connected to the same node. By contradiction, let us assume that the returned set of nodes is not a correct vertex cover: there exists an edge (u, v) in G whose two connected machines are not selected. By construction of G', there exists an edge between u_1 and v_2, and an edge between u_2 and v_1. Since neither u nor v are selected in G then, in the matching in G', no edge is connected to u_1, u_2, v_1 or v_2. As a result, it would be possible to add the edges (u_1, v_2) and (u_2, v_1) without breaking the matching, which contradicts our hypothesis of a maximal matching. As a result, one of our initial hypotheses was false. Since we know for sure that the matching is maximal, it must be that the returned set of nodes is a correct vertex cover. □

Theorem 1.24. *Algorithm 1.21 returns at most a 4-approximation of the minimum vertex cover in $O(\Delta)$ rounds in the port numbering model.*

Proof. Recall that the endpoints of a maximal matching form a 2-approximate vertex cover. Because any vertex cover of G induces a vertex cover of G' of at

most twice the size, the returned set has at most 4 times the size of a minimum vertex cover. As all edges in G' have at least one incident node in the vertex cover of G', the same is true in the computed node set of G, i.e., we did indeed find a vertex cover of G. □

Remarks:

- The term $O()$ used in Theorem 1.24 is called "big O" and is often used in distributed computing. Roughly speaking, $O(f)$ means "in the order of f, ignoring constant factors and smaller additive terms." More formally, for two functions f and g, it holds that $f \in O(g)$ if there are constants x_0 and c so that $|f(x)| \leqslant c|g(x)|$ for all $x \geqslant x_0$. For an elaborate discussion on the big O notation, we refer to other introductory math or computer science classes, or Wikipedia.
- If we look a bit closer, there's another surprise. The result is, in fact, a 3-approximation!

Corollary 1.25. *Algorithm 1.21 returns a 3-approximation of the minimum vertex cover.*

Proof. Consider the originals of the matching edges in the maximal matching of G'. These induce a subgraph of G of maximum degree 2, which is a disjoint union of paths and cycles of $k \geqslant 2$ nodes. To cover all edges of G, one needs to cover in particular these edges, and they can only be covered by the nodes on the respective paths and cycles. To cover a cycle of k nodes, at least $k/2$ of its nodes must be selected. To cover a path of k nodes, at least $\lfloor k/2 \rfloor$ of its nodes must be selected. An example is shown in Figure 1.26 As we choose all these nodes (and no others), the size of the computed vertex cover is at most factor $3 = \max_{2 \leqslant k \in \mathbb{N}} \{k/\lfloor k/2 \rfloor\}$ larger than the optimum. □

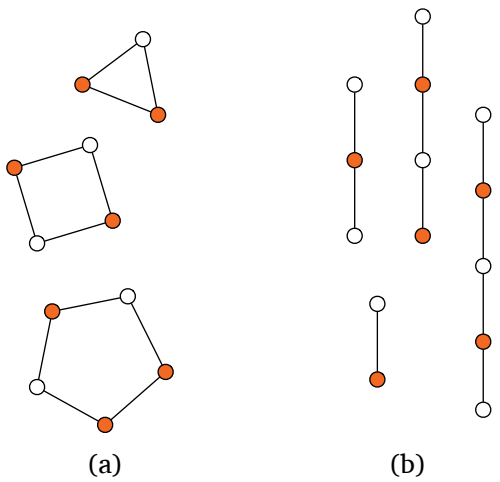

Figure 1.26: Examples for minimum vertex covers of cycles (a) and paths (b).

Remarks:

- In summary, the minimum vertex cover algorithm finds a 3-approximation of a minimum vertex cover in any graph G. Moreover, if the maximum degree of G is small, the algorithm is fast: we only need $O(\Delta)$ rounds in a network of maximum degree Δ.

Chapter Notes

The concept of port numbering comes from Angluin's [Ang80] work. The bipartite maximal matching algorithm is due to Hańćkowiak et al. [HKP98], and the minimum vertex cover 3-approximation algorithm is due to Polishchuk and Suomela [PS09]. These lecture notes are based on the lecture notes of Juho Hirvonen and Jukka Suomela [HS23], as well as the lecture notes of Christoph Lenzen [Len19].

Bibliography

[Ang80] Dana Angluin. Local and global properties in networks of processors. In *Proc. 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, 1980.

[HKP98] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. In *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*, 1998.

[HS23] Juho Hirvonen and Jukka Suomela. Distributed algorithms. `https://jukkasuomela.fi/da2020/`, October 2023.

[Len19] Christoph Lenzen. Theory of distributed systems. `https://www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/winter19/tods`, 2019.

[PS09] Valentin Polishchuk and Jukka Suomela. A simple local 3-approximation algorithm for vertex cover. *Information Processing Letters*, 109(12):642–645, 2009.