Joint Base Station Scheduling

Joint Base Station Scheduling*

Thomas Erlebach † Riko Jacob ‡ Matúš Mihaľák § Marc Nunkesser‡
Gábor Szabó‡ Peter Widmayer‡

Abstract

Consider a scenario where radio base stations need to send data to users with wireless devices. Time is discrete and slotted into synchronous rounds. Transmitting a data item from a base station to a user takes one round. A user can receive the data item from any of the base stations. The positions of the base stations and users are modeled as points in Euclidean space. If base station b transmits to user u in a certain round, no other user within distance at most $\|b-u\|_2$ from b can receive data in the same round due to interference phenomena. The goal is to minimize, given the positions of the base stations and users, the number of rounds until all users have received their data.

We call this problem the Joint Base Station Scheduling Problem (JBS) and consider it on the line (1D-JBS) and in the plane (2D-JBS). For 1D-JBS, we give an efficient 2-approximation algorithm and polynomial time optimal algorithms for special cases. We model transmissions from base stations to users as arrows (intervals with a distinguished endpoint) and show that their conflict graphs, which we call arrow graphs, are a subclass of the class of perfect graphs.

For 2D-JBS, we prove *NP*-hardness and show that some natural greedy heuristics do not achieve approximation ratio better than $\mathcal{O}(\log n)$, where n is the number of users.

1 Introduction

We consider different combinatorial aspects of problems that arise in the context of load balancing in time division networks. These problems turn out to be related to interval scheduling problems and interval graphs.

The general setting is that users with mobile devices are served by a set of base stations. In each time slot (round) of the time division multiplexing each base station serves at most one user. Traditionally, each user is assigned to a single base station that serves him until he leaves its cell or his demand is satisfied. The amount of data that a user receives depends on the strength of the signal that he receives from his assigned base station and on the interference, i.e. all signal power that he receives from other base stations. In [4], Das et al. propose a novel approach: Clusters of base stations jointly decide which users they serve in which round in order to increase network performance. Intuitively, this approach increases throughput, when in each round neighboring base stations try to serve pairs of users such that the mutual interference is low. We turn this approach into a discrete scheduling problem in one and two dimensions (see Figure 1.1), the Joint Base Station Scheduling problem (JBS).

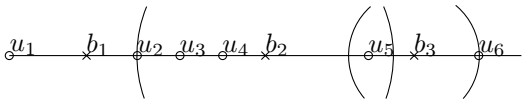
In one dimension (see Figure 1.1(a)) we are given a set of n users as points $\{u_1, \ldots, u_n\}$ on a line and we are given positions $\{b_1, \ldots, b_m\}$ of m base stations. Note that such a setting could correspond to a scenario where the base stations and users are located along a straight road. In our model, when a base station b_j serves a user u_i this creates interference in an interval of length $2|b_j - u_i|$ around the midpoint b_j. In each round each base station can serve at most one user such that at the position of this user there is no interference from any other base station. The goal is to serve all users in as few rounds as possible. In two dimensions users and base stations are represented as points in the plane. When base station b_j serves user u_i this creates interference in a disk with radius $\|b_j - u_i\|_2$ and center b_j (see Figure 1.1(c)).

*Research partially supported by TH-Project TH-46/02-1 (Mobile phone antenna optimization: theory, algorithms, engineering, and experiments).

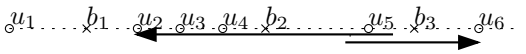
†Department of Computer Science, University of Leicester, t.erlebach@mcs.le.ac.uk

‡Department of Computer Science, ETH Zürich, {rjacob,nunkesser,szabog,widmayer}@inf.ethz.ch

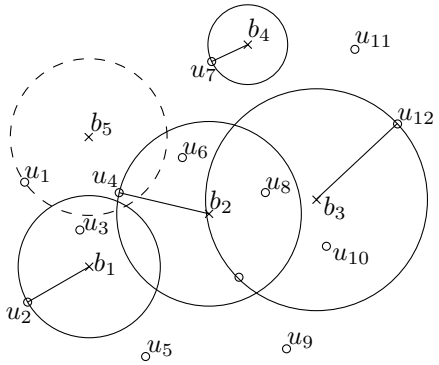
§Department of Information Technology and Electrical Engineering, ETH Zürich, mihalak@tik.ee.ethz.ch



(a) This figure describes a possible situation in some time slot (round). Base station b_2 serves user u_2, b_3 serves user u_6. Users u_3, u_4 and u_5 are blocked and cannot be served. Base station b_1 cannot serve u_1 because this would create interference at u_2



(b) Arrow representation of (a)



(c) A possible situation in some time slot in the 2D case. Users u_2, u_4, u_7 and u_{12} are served. Base station b_5 cannot serve user u_1 here, because this would create interference at u_4 as indicated by the dashed circle

Figure 1.1: The JBS-problem in one and two dimensions

The one-dimensional problem is closely related to interval scheduling problems, except that the particular way how interference operates leads to directed intervals (arrows). For these we allow that their tails can intersect (intersecting tails correspond to interference that does not affect the users at the heads of the arrows). We present results on this special interval scheduling problem. Similarly, the problem is related to interval graphs, except that we have conflict graphs of arrows together with the conflict rules defined by the interference (arrow graphs).

The problem of scheduling data transmissions in the smallest number of discrete rounds can be expressed as the problem of coloring the corresponding arrow graph with the smallest number of colors, where the colors represent rounds. In this paper, we prove that arrow graphs are perfect and can be colored optimally in $\mathcal{O}(n \log n)$ time. For the one-dimensional JBS problem with evenly spaced base stations we give a polynomial-time dynamic programming algorithm. For another special case of the one-dimensional JBS problem, where $3k$ users must be served by 3 base stations in k rounds, we also give a polynomial-time optimal algorithm. For the general one-dimensional JBS problem, we show that for any fixed k the question whether all users can be served in k rounds can be solved in $n^{\mathcal{O}(k)}$ time. From the perfectness of arrow graphs and the existence of a polynomial-time algorithm for computing maximum weighted cliques in these graphs we derive a 2-approximation algorithm for JBS based on an LP relaxation and rounding. We show that this result can also be generalized to a more realistic model where the interference region extends beyond the receiver. For the two-dimensional JBS problem, we show that it is *NP*-complete, while deciding whether all users can be served in one round can be done in polynomial time. We analyze an approximation algorithm for a constrained version of the problem, and present lower bounds on the quality of some natural greedy algorithms for the general two-dimensional JBS problem.

1.1 Related Work

Das et al. [4] propose an involved model for load balancing that takes into account different fading effects and calculates the resulting signal to noise ratios at the users for different schedules. In each round only a subset of all base stations is used in order to keep the interference low. The decision which base stations to use is taken by a central authority. The search for this subset is formulated as a (nontrivial) optimization problem that is solved by complete enumeration and that assumes complete knowledge of the channel conditions. The authors perform simulations on a hexagonal grid, propose other algorithms, and reach the conclusion that the approach has the potential to increase throughput.

There is a rich literature on interval scheduling and selection problems (see [6, 12] and

the references given there for an overview). Our problem is more similar to a setting with several machines where one wants to minimize the number of machines required to schedule all intervals. A version of this problem where intervals have to be scheduled within given time windows is studied in [3]. Inapproximability results for the variant with a discrete set of starting times for each interval are presented in [2].

1.2 Problem Definitions and Model

We fully define the problems of interest in this section. Throughout the paper we use standard graph-theoretic terminology, see e.g. [14]. In the one-dimensional case we are given positions of base stations $B = \{b_1, \ldots, b_m\}$ and users $U = \{u_1, \ldots, u_n\}$ on the line in left-to-right order. Conceptually, it is more convenient to think of the interference region that is caused by some base station b_j serving a user u_i as an *interference arrow* of length $2|b_j - u_i|$ with midpoint b_j pointing to the user as shown in Figure 1.1(b). The interference arrow for the pair (u_i, b_j) has its head at u_i and its midpoint at b_j. We denote the set of all arrows resulting from pairs $P \subseteq U \times B$ by $\mathcal{A}(P)$. If it is clear from the context, we call the interference arrows just *arrows*. If two users are to be scheduled in the same round, then each of them must not get any interference from any other base station. Thus, two arrows are *compatible* if no head is contained in the other arrow; otherwise, we say that they are in *conflict*. (Formally, the head u_i of the arrow for (u_i, b_j) is contained in the arrow for (u_j, b_k) if u_i is contained in the closed interval $[b_k - |u_j - b_k|, b_k + |u_j - b_k|]$.) If we want to emphasize which user is affected by the interference from another transmission, we use the term *blocking*, i.e. arrow a_i blocks arrow a_j if a_j's head is contained in a_i. For each user we have to decide from which base station she is served. This corresponds to a selection of an arrow for her. Furthermore, we have to decide in which round each selected arrow is scheduled under the side constraint that all arrows in one round must be compatible. For this purpose it is enough to label the arrows with colors that represent the rounds.

For the two-dimensional JBS problem we have positions in $\mathbb{R}^2$ and *interference disks* $d(b_i, u_j)$ with center b_i and radius $\|b_i - u_j\|_2$ instead of arrows. We denote the set of interference disks for the user base-station pairs from a set P by $\mathcal{D}(P)$. Two interference disks are in conflict if the user that is served by one of the disks is contained in the other disk; otherwise, they are compatible. The problems can now be stated as follows:

1D-JBS

Input: User positions $U = \{u_1, \ldots, u_n\} \subset \mathbb{R}$ and base station positions $B = \{b_1, \ldots, b_m\} \subset \mathbb{R}$.

Output: A set P of n user base-station pairs such that each user is in exactly one pair, and a coloring $C : \mathcal{A}(P) \to \mathbb{N}$ of the set $\mathcal{A}(P)$ of corresponding arrows such that any two arrows $a_i, a_j \in \mathcal{A}(P),\ a_i \neq a_j$, with $C(a_i) = C(a_j)$ are compatible.

Objective: Minimize the number of colors used.

2D-JBS

Input: User positions $U = \{u_1, \ldots, u_n\} \subset \mathbb{R}^2$ and base station positions $B = \{b_1, \ldots, b_m\} \subset \mathbb{R}^2$.

Output: A set P of n user base-station pairs such that each user is in exactly one pair, and a coloring $C : \mathcal{D}(\mathcal{P}) \to \mathbb{N}$ of the set $\mathcal{D}(\mathcal{P})$ of corresponding disks such that any two disks $d_i, d_j \in \mathcal{D}(\mathcal{P}),\ d_i \neq d_j$, with $C(d_i) = C(d_j)$ are compatible.

Objective: Minimize the number of colors used.

For simplicity we will write c_i instead of $C(a_i)$ in the rest of the paper. From the problem definitions above it is clear that both the 1D- and the 2D-JBS problems consist of a *selection problem* and a *coloring problem*. In the selection problem we want to select one base station for each user in such a way that the arrows (disks) corresponding to the resulting set P of user base-station pairs can be colored with as few colors as possible. We call a selection P *feasible* if it contains exactly one user base-station pair for each user. Determining the cost of a selection is then the coloring problem. This can also be viewed as a problem in its own right, where we no longer make any assumption on how the set of arrows (for the 1D problem) is produced.

The conflict graph $G(A)$ of a set A of arrows is the graph in which every vertex corresponds to an arrow and there is an edge between two vertices if the corresponding arrows are in conflict. We call such conflict graphs of arrows *arrow graphs*. The *arrow graph coloring problem* asks for a proper coloring of such a graph. It is similar in spirit to the coloring of interval graphs. As we will see in Section 2.1, the arrow graph coloring problem can be solved in time $\mathcal{O}(n \log n)$. We finish this section with a simple lemma that leads to a definition:

Lemma 1.1. *For each 1D-JBS instance there is an optimal solution in which each user is served either by the closest base station to her left or by the closest base station to her right.*

Proof. This follows by a simple exchange argument: Take any optimal solution that does not have this form. Then exchange the arrow where a user is not served by the closest base station in some round against the arrow from the closest base station on the same side (which must be idle in that round). Shortening an arrow without moving its head can only resolve conflicts. Thus, there is also an optimal solution with the claimed property. □

The two possible arrows by which a user can be served according to this lemma are called *user arrows*. It follows that for a feasible selection one has to choose one user arrow from each pair of user arrows.

2 Case on the Line—1D-JBS

As mentioned above, solving the 1D-JBS problem requires selecting an arrow for each user and coloring the resulting arrow graph with as few colors as possible. Trying to understand when a selection of arrows leads to an arrow graph with small chromatic number, we first study the coloring problem for arrow graphs.

2.1 Coloring Arrow Graphs

In this section we consider the arrow graph coloring problem: Given a set $A = \{a_1, \ldots, a_n\}$ of interference arrows, color it with the minimum number of colors in such a way that no two arrows of the same color are in conflict. We present a simple greedy algorithm that colors $G(A)$ optimally in time $\mathcal{O}(n \log n)$. It is a modification of the standard greedy algorithm [14] for coloring interval graphs. We remark that the existence of an $\mathcal{O}(n \log n)$ coloring algorithm for arrow graphs follows also from the fact that arrow graphs are a special case of trapezoid graphs (cf. Section 2.3). The algorithm we describe in the following can be viewed as an adaptation of the coloring algorithm for trapezoid graphs due to Felsner et al. [7]. We present the algorithm here to make the paper more self-contained and because the description and analysis of the algorithm may be helpful to better understand the properties of arrow graphs.

We assume for simplicity that the arrows are given in left-to-right order of their left endpoints. This sorting can also be seen as the first step of the algorithm. Then the algorithm scans the arrows from left to right in this sorted order. In step i it checks whether there are colors that have already been used and that can be assigned to a_i without creating a conflict. If there are such candidate colors, it considers for each such color c the rightmost right endpoint r_c among the arrows that have been assigned color c so far, and chooses for a_i a color c for which r_c is rightmost (breaking ties arbitrarily). If there is no candidate color, the algorithm assigns a new color to a_i.

We show that this greedy algorithm produces an optimal coloring by showing that any optimal solution can be transformed into the solution produced by the algorithm.

Lemma 2.1. *Let C be an optimal coloring for a set of arrows $A = \{a_1, \ldots, a_n\}$. The coloring C can be transformed into the coloring produced by the greedy algorithm without introducing new colors.*

Proof. We show the lemma by induction on the index of the arrows. The induction hypothesis is: *There exists an optimal coloring that agrees with the greedy coloring up to arrow $k-1$.* The induction start is trivial. In the kth step let $C = (c_1, \ldots, c_n)$ be such an optimal coloring and let $H = (h_1, \ldots, h_n)$ be the greedy coloring, i.e. we have $h_1 = c_1, h_2 = c_2, \ldots, h_{k-1} = c_{k-1}$. We consider

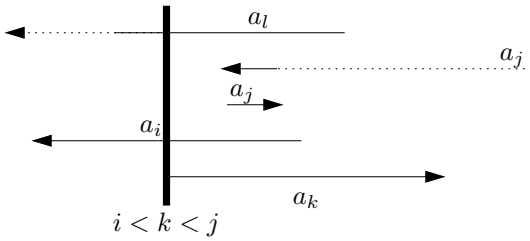


Figure 2.1: Possible configuration for the two cases. Dotted lines mean that the arrows could be extended

the coloring $C' = (c'_1, \ldots, c'_n)$ that is obtained from C by exchanging the colors c_k and h_k for the arrows $a_k, \ldots, a_n$. More precisely, we define

$$c'_i = \begin{cases} c_i, & \text{if } i < k \text{ or } c_i \notin \{c_k, h_k\} \\ h_k, & \text{if } i \geq k \text{ and } c_i = c_k \\ c_k, & \text{if } i \geq k \text{ and } c_i = h_k. \end{cases}$$

By definition we have $c'_k = h_k$, and it remains to show that C' is a proper coloring and, therefore, the induction hypothesis is also true for k. If $c_k = h_k$ we have $C' = C$ which is a proper coloring. Otherwise, we have to show that all pairs of arrows a_i, a_j that are in conflict receive different colors in C', i.e., $c'_i \neq c'_j$. If $i, j < k$ or $k \leq i, j$ this is obvious by the fact that C is a coloring. Hence, we assume $i < k < j$; the case $j = k$ is implied by H being a proper coloring.

If h_k is a new color, i.e. different from all of $c_1, \ldots, c_{k-1}$, then, because of the greedy algorithm, also c_k is a new color. Hence, it is impossible that we have $c'_i = c'_j$.

Now assume for a contradiction that we indeed have $c = c'_i = c'_j$ and the arrows a_i and a_j are in conflict. By the ordering of the arrows we know that a_i and a_k overlap. Observe that $c \in \{c_k, h_k\}$ because C is a coloring. This leaves us with two cases:

Case 1 $c = c_k$**:** Since C is a coloring, the arrows a_i and a_k are compatible, i.e. a_i is directed left and a_k is directed right. Such a configuration is depicted in Figure 2.1. By the definition of the greedy algorithm, we know that h_k is a color of a compatible arrow. Since $h_k \neq c_k = c_i$, there must exist an arrow a_l, $l < k$, that ends not before a_i and has color h_k, i.e. $c_l = h_k$ (and a_l is compatible with a_k). Since a_j is in conflict with a_i (the head of a_j is within a_i), there is also a conflict between a_j and a_l. We have $c'_j = c_k$, implying $c_j = h_k$, hence we get the contradiction $c_j = h_k = c_l$ in the optimal coloring C.

Case 2 $c = h_k$**:** Because H is a coloring, a_i and a_k have to be compatible. Since a_i ends before a_k and is in conflict with a_j, also a_j is in conflict with a_k. Because $c'_j = h_k$ we know by definition of C' that $c_j = c_k$, hence there is a conflict in C, a contradiction. □

The running time of the algorithm depends on the time the algorithm spends in every step on identifying an allowed color that was previously assigned to an arrow with the rightmost right endpoint. By maintaining two balanced search trees (one tree for each direction of arrows) storing the most recently colored arrows of the used colors (one arrow per color) in the order of their right endpoints, we can implement this operation in logarithmic time. Together with Lemma 2.1 we get the following theorem.

Theorem 2.2. *The greedy algorithm optimally colors a given set of arrows* $\{a_1, \ldots, a_n\}$ *in* $\mathcal{O}(n \log n)$ *time.*

2.2 1D-JBS with Evenly Spaced Base Stations

Now consider the 1D-JBS problem under the assumption that the base stations are evenly spaced. We are given m base stations $\{b_1, \ldots, b_m\}$ and n users $\{u_1, \ldots, u_n\}$ on a line, where the distance between any two neighboring base stations is the same. This assumption can be viewed as an abstraction of the fact that in practice, base stations are often placed in regular patterns and not in a completely arbitrary fashion.

Let d denote the distance between two neighboring base stations. The base stations partition the line into two *rays* and a set of *intervals* $\{v_1, \ldots, v_{m-1}\}$. In this section we additionally require

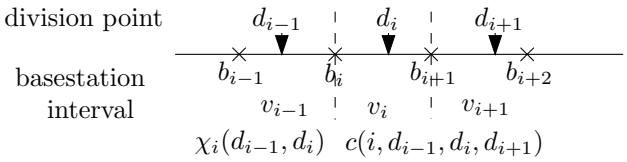


Figure 2.2: Dynamic programming approach

that no user to the left of the leftmost base station be further away from it than distance d, and that the same hold for the right end. We define a solution to be *non-crossing* if there are no two users u and w in the same interval such that u is to the left of w, u is served from the right, and w from the left.

Lemma 2.3. *For instances of 1D-JBS with evenly spaced base stations, there is always an optimal solution that is non-crossing.*

Proof. Take any optimal solution s that is not non-crossing. We show that such a solution can be transformed into another optimal solution s' that is non-crossing. Let u and w be two users such that u and w are in the same interval, u is to the left of w, and u is served by the right base station b_r in round t_1 by arrow a_r and w is served by the left base station b_l in round t_2 by arrow a_l; trivially, $t_1 \neq t_2$. Modify s in such a way that at t_1 base station b_r serves w and at t_2 base station b_l serves u. This new solution is still feasible because first of all both the left and the right involved arrows a_l and a_r have become shorter. This implies that both a_l and a_r can only block fewer users. On the other hand, the head of a_l has moved left and the head of a_r has moved right. It is impossible that they are blocked now because of this movement: In t_1 this could only happen if there were some other arrows containing w, the new head of a_r. Such an arrow cannot come from the left, because then it would have blocked also the old arrow. It cannot come from b_r because b_r is busy. It cannot come from a base station to the right of b_r, because such arrows do not reach any point to the left of b_r (here we use the assumption that the rightmost user is no farther to the right of the rightmost base station than d, and that the base stations are evenly spaced). For t_2 the reasoning is symmetric. □

The selection of arrows in any non-crossing solution can be completely characterized by a sequence of $m-1$ *division points*, such that the i^{th} division point is the index of the last user that is served from the left in the i^{th} interval. (The case where all users in the i^{th} interval are served from the right is handled by choosing the i^{th} division point as the index of the rightmost user to the left of the interval, or as 0 if no such user exists.) A brute-force approach could now enumerate over all possible $\mathcal{O}(n^{m-1})$ division point sequences (*dps*) and color the selection of arrows corresponding to each dps with the greedy algorithm.

Dynamic Programming

We can solve the 1D-JBS problem with evenly spaced base stations more efficiently by a dynamic programming algorithm that runs in polynomial time. The idea of the algorithm is to consider the base stations and thus the intervals in left-to-right order. We consider the cost $\chi_i(d_{i-1}, d_i)$ of an optimal solution up to the ith base station conditioned on the position of the division points d_{i-1} and d_i in the intervals v_{i-1} and v_i, respectively, see Figure 2.2.

Definition 2.4. We denote by $\chi_i(\alpha, \beta)$ the minimum number of colors needed to serve users u_1 to u_β using the base stations b_1 to b_i under the condition that base station b_i serves exactly users $u_{\alpha+1}$ to u_β and ignoring the users $u_{\beta+1}, \ldots, u_n$.

Let $\Lambda(v_i)$ denote the set of potential division points for interval v_i, i.e., the set of the indices of users in v_i and of the rightmost user to the left of v_i (or 0 if no such user exists). The values $\chi_1(d_0, d_1)$ for $d_0 = 0$ (all users to the left of b_1 must be served by b_1 in any solution) and $d_1 \in \Lambda(v_1)$ can be computed directly by using the greedy coloring algorithm. For $i \geq 1$, we compute the values $\chi_{i+1}(d_i, d_{i+1})$ for $d_i \in \Lambda(v_i)$, $d_{i+1} \in \Lambda(v_{i+1})$ from the table for $\chi_i(\cdot, \cdot)$. If we additionally fix a division point d_{i-1} for interval v_{i-1}, we know exactly which selected arrows intersect interval v_i regardless of the choice of other division points. Observe that this only holds for equidistant base stations and no "far out" users. For this selection, we can determine with

the greedy coloring algorithm of Section 2.1 how many colors are needed to color the arrows intersecting v_i. Let us call this number $c(i, d_{i-1}, d_i, d_{i+1})$ for interval v_i and division points d_{i-1}, d_i and d_{i+1}. We also know how many colors we need to color the arrows intersecting intervals v_0 to v_{i-1}. For a fixed choice of division points d_{i-1}, d_i and d_{i+1} we can combine the two colorings corresponding to $\chi_i(d_{i-1}, d_i)$ and $c(i, d_{i-1}, d_i, d_{i+1})$: Both of these colorings color all arrows of base station b_i, and these arrows must all have different colors in both colorings. No other arrows are colored by both colorings, so $\chi_i(d_{i-1}, d_i)$ and $c(i, d_{i-1}, d_i, d_{i+1})$ agree up to redefinition of colors. We can choose the best division point d_{i-1} and get

$$\chi_{i+1}(d_i, d_{i+1}) = \min_{d_{i-1} \in \Lambda(v_{i-1})} \max\{\chi_i(d_{i-1}, d_i), c(i, d_{i-1}, d_i, d_{i+1})\}$$

The running time is dominated by the calculation of the $c(\cdot)$ values. There are $\mathcal{O}(m \cdot n^3)$ such values, and each of them can be computed in time $\mathcal{O}(n \log n)$ using the coloring algorithm. The optimal solution can be found in the usual way by tracing back where the minimum was achieved from $\chi_m(x, n)$. Here the x is chosen among the users of the interval before the last base station such that $\chi_m(x, n)$ is minimum. For the traceback it is necessary to store in the computation of the χ values where the minimum was achieved. The traceback yields a sequence of division points that defines the selection of arrows that gives the optimal schedule. Altogether, we have shown the following theorem:

Theorem 2.5. *The base station scheduling problem for evenly spaced base stations can be solved in time $\mathcal{O}(m \cdot n^4 \log n)$ by dynamic programming.*

Note that the running time can also be bounded by $\mathcal{O}(m \cdot u_{\max}^4 \log u_{\max})$, where $u_{\max}$ is the maximum number of users in one interval.

2.3 Relation to Other Graph Classes and Perfectness

We have shown in Section 2.1 that optimal colorings of arrow graphs can be computed efficiently. We will make use of additional properties of arrow graphs in the design of algorithms for 1D-JBS in the following sections. Therefore, let us briefly discuss the relationship between arrow graphs and other known graph classes.[1] We refer to [1, 13] for definitions and further information about the graph classes mentioned in the following.

First, it is easy to see that arrow graphs are a superclass of interval graphs: Any interval graph can be represented as an arrow graph with all arrows pointing in the same direction.

An arrow graph can be represented as the intersection graph of triangles on two horizontal lines $y = 0$ and $y = 1$: Simply represent an arrow with left endpoint ℓ and right endpoint r that points to the right (left) as a triangle with corners $(\ell, 0)$, $(r, 0)$, and $(r, 1)$ (with corners $(r, 1)$, $(\ell, 1)$, and $(\ell, 0)$). It is easy to see that two triangles intersect if and only if the corresponding arrows are in conflict. See Figure 2.3 for an example.

Intersection graphs of triangles with endpoints on two parallel lines are called PI* graphs. They are a subclass of trapezoid graphs, which are the intersection graphs of trapezoids that have two sides on two fixed parallel lines. Trapezoid graphs are in turn a subclass of co-comparability graphs, a well-known class of perfect graphs. Therefore, the containment in these known classes of perfect graphs implies the perfectness of arrow graphs. Consequently, the size of a maximum clique in an arrow graph always equals its chromatic number.

As arrow graphs are a subclass of trapezoid graphs, we can apply known efficient algorithms for trapezoid graphs to arrow graphs. Felsner et al. [7] give algorithms with running-time $\mathcal{O}(n \log n)$ for chromatic number, weighted independent set, clique cover, and weighted clique in trapezoid graphs with n vertices, provided that the trapezoid representation is given. The coloring algorithm provided there is very similar to the algorithm explained in Section 2.1. We sum up the discussed properties of arrow graphs in the following theorem.

Theorem 2.6. *Arrow graphs are perfect. In arrow graphs chromatic number, weighted independent set, clique cover, and weighted clique can be solved in time $\mathcal{O}(n \log n)$.*

One can also show that arrow graphs are AT-free (i.e., do not contain an asteroidal triple) and weakly chordal.

[1] The connections between arrow graphs and known graph classes such as PI* graphs, trapezoid graphs, co-comparability graphs, AT-free graphs, and weakly chordal graphs were observed by Ekki Köhler, Jeremy Spinrad, Ross McConnell, and R. Sritharan at the seminar "Robust and Approximative Algorithms on Particular Graph Classes", held in Dagstuhl Castle during May 24–28, 2004.

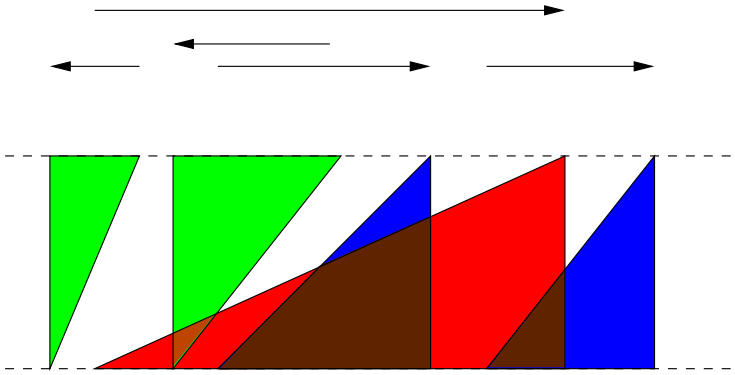


Figure 2.3: An arrow graph (top) and its representation as a PI* graph (bottom)

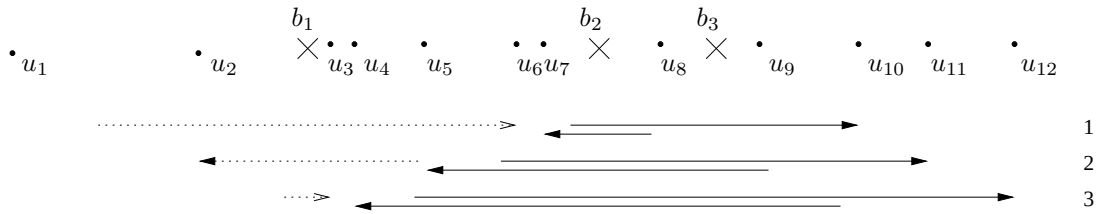


Figure 2.4: Far away users u_{10}, u_{11} and u_{12} are served by b_3 in rounds 1, 2 and 3, respectively. The solid arrows depict the selection of users for b_2 and b_3. The dotted arrows depict the resulting selection for b_1. Users u_1, u_8 and u_9 will be scheduled in a round of Type 2 (not shown).

2.4 Serving $3k$ Users with 3 Base Stations in k Rounds

We study another special case of the general setting. We are given 3 base stations b_1, b_2 and b_3, and $3k$ users with k *far away* users among them, i.e., users to the left of b_1 or to the right of b_3 whose interference arrows contain b_2; the problem is to decide whether the users can be served in k rounds.

Observe that in every round every base station has to serve a user. We know that a far away user has to be served by its unique neighboring base station. Since the arrows of far away users contain b_2, all users between b_1 and b_2 have to be served in rounds in which far away users of b_3 are served, and all users between b_2 and b_3 have to be served in rounds in which far away users of b_1 are served. In particular, every round must be of one of the following two types:

Type 1: b_3 serves a far away user, b_2 serves a user between b_1 and b_2, and b_1 serves a user that is not a far away user.

Type 2: b_1 serves a far away user, b_2 serves a user between b_2 and b_3, and b_3 serves a user that is not a far away user.

For every user, it is uniquely determined whether she will be served in a round of Type 1 or Type 2.

The schedule can be constructed in the following way. Suppose we have k_1 far away users at b_1 and k_3 far away users at b_3, $k = k_1 + k_3$. First, we serve the far away users of b_3 in rounds $1, \ldots, k_3$ in the order of increasing distance from b_3. Next, we match the resulting arrows with arrows produced by b_2 serving users between b_1 and b_2 (cf. Figure 2.4). We apply a best fit approach. For every round $i = 1, 2, \ldots, k_3$, we find the user closest to b_2 that can be served together with the corresponding far away user served by b_3, and schedule the corresponding transmission in that round. Note that with this selection the size of the arrows of b_2 grows with the number of the round in which they are scheduled. Now we have to serve the remaining k_3 users (that are not far away users of b_1) with b_1. We use a best fit approach again, i.e., for every round $i = 1, 2, \ldots, k_3$, we schedule the user with maximum distance from b_1 (longest arrow) among the remaining users. This completes the description of how we obtain a schedule for the rounds in which a far away user of b_3 is served. The schedule for the remaining users (the

far away users of b_1, and the users that must be scheduled in a round in which a far away user of b_1 is scheduled) can be found similarly, starting with the far away users of b_1.

We claim that if there exists a valid schedule with k rounds for the given instance of the problem, our algorithm produces a valid schedule. Without loss of generality, we consider only the schedule for the rounds in which the far away users of b_3 are served; the reasoning for the rounds with far away users of b_1 is analogous. Consider any valid schedule with k_3 rounds for the $3k_3$ users that must be served in rounds in which far away users of b_3 are served; call this schedule the *optimal schedule*. We can assume that the far away users are served by b_3 in the optimal schedule in the same round as in our schedule (the schedule produced by our algorithm). We will show that we can transform the optimal schedule into our schedule without losing validity.

First, we transform the optimal schedule in such a way that, in addition to the user served by b_3, also the user served by b_2 is the same as in our schedule in every round. Consider the first round i in which the optimal schedule does not serve the same user with b_2 as our schedule. Assume that b_2 serves user x in our schedule, but user $y \neq x$ in the optimal schedule. (Note also that the algorithm cannot get stuck while selecting a user to be served by b_2 in round i, since y is a candidate.) Note that y must be to the left of x, due to the best-fit rule our algorithm applies. The optimal schedule must serve x in some round $j \neq i$. If it serves x with b_2, we know that $j > i$, and we can simply exchange the users served by b_1 and b_2 in rounds i and j in the optimal schedule. If the optimal schedule serves x with b_1 in round j, we can let the optimum serve y with b_1 in round j and x with b_2 in round i. In both cases, we have transformed the optimal schedule so that it also serves x with b_2 in round i, without losing validity. Repeating this transformation, we obtain an optimal schedule that seves the same users with b_2 and b_3 as our schedule in every round.

It remains to handle the users served by b_1. Consider the first round i in which the optimal schedule differs from our schedule. Assume that b_1 serves user w in our schedule, but user z in the optimal schedule. Note that, by the best-fit approach, the arrow for (w, b_1) must be at least as long as the arrow for (z, b_1). The optimal schedule must serve w in some later round, say, round j, also with b_1. We can change the optimal schedule by letting b_1 serve user w in round i and user z in round j; this does not affect the validity of the schedule, since the arrow for z is not longer than the arrow for w (and thus round j remains valid) and since the arrow produced by the transmission of b_2 is shorter in round i than in round j (and thus serving w in round i must be valid if serving w in round j was valid). By repeating this transformation, we have transformed the optimal schedule into our schedule, without losing validity. This shows that our algorithm produces a valid schedule if one exists. The time complexity of our algorithm is dominated by the time for sorting the users, $\mathcal{O}(n \log n)$. After the sorting, the schedule can be computed in linear time.

Theorem 2.7. *For the setting with 3 base stations and $3k$ users on a line with k far away users, there is an $\mathcal{O}(n \log n)$ time algorithm that computes a valid schedule with k rounds if there is one.*

2.5 Exact Algorithm for the k-Decision Problem

In this section we present an exact algorithm for the decision variant k-1D-JBS of the 1D-JBS problem: For given k and an instance of 1D-JBS, decide whether all users can be served in at most k rounds. We present an algorithm for this problem that runs in $\mathcal{O}(m \cdot n^{2k+1} \log n)$ time.

We use the result from Section 2.3 that arrow graphs are perfect. Thus the size of the maximum clique of an arrow graph equals its chromatic number.

The idea of the algorithm, which we call $A_{k-\mathrm{JBS}}$, is to divide the problem into subproblems, one for each base station, and then combine the partial solutions to a global one.

For base station b_i, the corresponding subproblem S_i considers only arrows that intersect b_i and arrows for which the alternative user arrow[2] intersects b_i. Call this set of arrows A_i. We call S_{i-1} and S_{i+1} *neighbors* of S_i. A solution to S_i consists of a feasible selection of arrows from A_i of cost no more than k, i.e. the selection can be colored with at most k colors. To find all such solutions we enumerate all possible selections that can lead to a solution in k rounds. For S_i we store all such solutions $\{s_i^1, \ldots, s_i^I\}$ in a table T_i. We only need to consider selections in which at most $2k$ arrows intersect the base station b_i. All other selections need more than

[2]For every user there are only two user arrows that we need to consider (Lemma 1.1). If we consider one of them, the other one is the *alternative user arrow*.

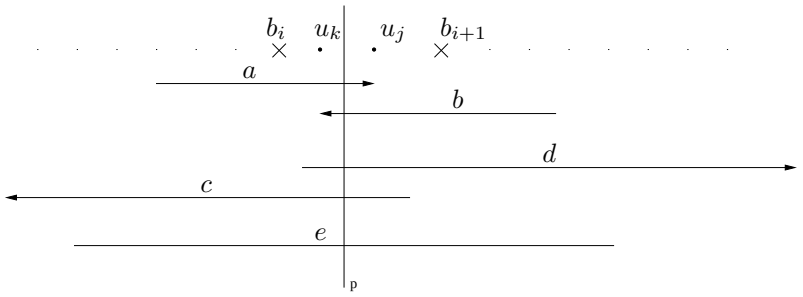


Figure 2.5: A clique of size greater than k at point p together with the different types of arrows at this point

k rounds, because they must contain more than k arrows pointing in the same direction at b_i. Therefore, the number of entries of T_i is bounded by $\sum_{j=0}^{2k} \binom{n}{j} = \mathcal{O}(n^{2k})$. We need $\mathcal{O}(n \log n)$ time to evaluate a single selection with the coloring algorithm of Section 2.1. Selections that cannot be colored with at most k colors are marked as irrelevant and ignored in the rest of the algorithm. We build up the global solution by choosing a set of feasible selections $s_1, \ldots, s_m$ in which all neighbors are compatible, i.e. they agree on the selection of common arrows. It is easy to see that in such a global solution all subsolutions are pairwise compatible.

We can find such a set of compatible neighbors by going through the tables in left-to-right order and marking every solution in each table as *valid* if there is a compatible, valid solution in the table of its left neighbor, or as *invalid* otherwise. A solution s_i marked as valid in table T_i thus indicates that there are solutions $s_1, \ldots, s_{i-1}$ in $T_1, \ldots, T_{i-1}$ that are compatible with it and pairwise compatible. In the leftmost table T_1, every feasible solution is marked as valid. When the marking has been done for the tables of base stations $b_1, \ldots, b_{i-1}$, we can perform the marking in the table T_i for b_i in time $\mathcal{O}(n^{2k+1})$ as follows. First, we go through all entries of the table T_{i-1} and, for each such entry, in time $\mathcal{O}(n)$ discard the part of the selection affecting pairs of user arrows that intersect only b_{i-1} but not b_i, and enter the remaining selection into an intermediate table $T_{i-1,i}$. The table $T_{i-1,i}$ stores entries for all selections of arrows from pairs of user arrows intersecting both b_{i-1} and b_i. An entry in $T_{i-1,i}$ is marked as valid if at least one valid entry from T_{i-1} has given rise to the entry. Then, the entries of T_i are considered one by one, and for each such entry s_i the algorithm looks up in time $\mathcal{O}(n)$ the unique entry in $T_{i-1,i}$ that is compatible with s_i to see whether it is marked as valid or not, and marks the entry in T_i accordingly. If in the end the table T_m contains a solution marked as valid, a set of pairwise compatible solutions from all tables exists and can be retraced easily.

The overall running time of the algorithm is $\mathcal{O}(m \cdot n^{2k+1} \cdot \log n)$. There is a solution to k-1D-JBS if and only if the algorithm finds such a set of compatible neighbors.

Lemma 2.8. *There exists a solution to k-1D-JBS if and only if $A_{k-\mathrm{JBS}}$ finds a set of pairwise compatible solutions.*

Proof. ($\Rightarrow$) Every arrow intersects at least one base station. A global solution directly provides us with a set of compatible subsolutions $\Sigma_{\mathrm{opt}} = \{s_1^{\mathrm{opt}}, \ldots, s_m^{\mathrm{opt}}\}$. Since the global solution has cost at most k, so have the solutions of the subproblems. Hence, the created entries will appear in the tables of the algorithm and will be considered and marked as valid. Thus there is at least one set of compatible solutions that is discovered by the algorithm.

($\Leftarrow$) We have to show that the global solution constructed from the partial ones has cost at most k. Suppose for a contradiction that there is a point p where the clique size is bigger than k and therefore bigger than the clique at b_i (the left neighboring base station of p) and the clique at b_{i+1} (the right neighboring base station of p). We divide the arrows intersecting point p into 5 groups as in Figure 2.5. Arrows of type a (b) have their head between b_i and b_{i+1} and their tail to the left (right) of b_i (b_{i+1}). Arrows of type c (d) have their tail between b_i and b_{i+1} and their head to the left (right) of b_i (b_{i+1}). Finally, type e arrows intersect both b_i and b_{i+1}. For the clique at p to be bigger than that at b_i some arrows not considered at b_i have to create conflicts. The only such arrows (considered at b_{i+1} but not at b_i) are of type d (observe that arrows of type a, b and e are considered both at the table for b_i and at the table for b_{i+1}). If their presence increases the clique size at p, then no type c arrow can be in the maximum clique at p (observe

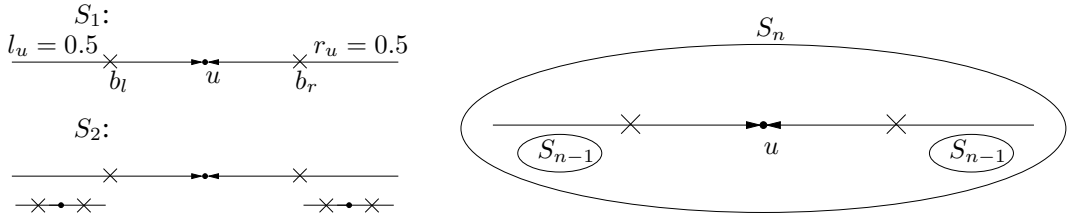


Figure 2.6: Setting S_1 with two base stations b_l and b_r and one user u in between, where both the solution of the ILP and the solution of the LP relaxation have cost 1. S_2 is constructed recursively by adding to the setting of S_1 two (scaled) copies of S_1 in the tail positions of the arrows. Here the cost of the relaxed LP is 1.5 and the integral cost is 2. The recursive approach for general n is shown in the right picture. Using setting S_1 and putting two (properly scaled) settings S_{n-1} as depicted in the picture, we get a setting S_n where $k^*(n)$, the cost of the LP relaxation for S_n, is $0.5 + k^*(n-1) = 0.5 + n/2$, whereas the cost of the ILP is n

that arrows of type c and d are compatible). Therefore, the clique at p cannot be bigger than the clique at b_{i+1}, a contradiction. □

Theorem 2.9. *Problem k-1D-JBS can be solved in time $\mathcal{O}(m \cdot n^{2k+1} \cdot \log n)$.*

2.6 Approximation Algorithm

In this section we present an approximation algorithm for 1D-JBS that relies on the properties of arrow graphs from Theorem 2.6. Let A denote the set of all user arrows of the given instance of 1D-JBS. From the perfectness of arrow graphs it follows that it is equivalent to ask for a feasible selection $A_{\text{sel}} \subseteq A$ minimizing the chromatic number of its arrow graph $G(A_{\text{sel}})$ (among all feasible selections) and to ask for a feasible selection A_{sel} minimizing the maximum clique size of $G(A_{\text{sel}})$ (among all feasible selections). Exploiting this equivalence, we can express the 1D-JBS problem as an integer linear program as follows. We introduce two indicator variables l_i and r_i for every user i that indicate whether she is served by the left or by the right base station, i.e. if the user's left or right user arrow is selected. Moreover, we ensure by the constraints that no cliques in $G(A_{\text{sel}})$ are large and that each user is served. The ILP formulation is as follows:

$$\begin{aligned}
\min \quad & k && (2.1)\\
\text{s.t.} \quad & \sum_{l_i \in C} l_i + \sum_{r_i \in C} r_i \leq k \qquad \forall \text{ cliques } C \text{ in } G(A) && (2.2)\\
& l_i + r_i = 1 \qquad \forall i \in \{1, \ldots, |U|\} && (2.3)\\
& l_i, r_i \in \{0,1\} \qquad \forall i \in \{1, \ldots, |U|\} && (2.4)\\
& k \in \mathbb{N} && (2.5)
\end{aligned}$$

The natural LP relaxation is obtained by allowing $l_i, r_i \in [0,1]$ and $k \geq 0$. Given a solution to this relaxation, we can use a rounding technique to get an assignment of users to base stations that has cost at most twice the optimum, i.e., we obtain a 2-approximation algorithm. Let us denote by *opt* the optimum number of colors needed to serve all users. Then *opt* $\geq k$, because the optimum integer solution is a feasible fractional solution. Construct now a feasible solution from a solution to the relaxed problem by rounding $l_i := \lfloor l_i + 0.5 \rfloor$, $r_i := 1 - l_i$. Before the rounding the size of every (fractional) clique is at most k; afterwards the size can double in the worst case (because the value of each individual variable can at most double). Therefore, the cost of the rounded solution is at most $2k \leq 2$*opt*. Figure 2.6 gives an example where the cost of an optimal solution to the relaxed program is indeed smaller than the cost of an optimal integral solution by a factor arbitrarily close to 2.

One issue that needs to be discussed is how the relaxation can be solved in time polynomial in n and m, as there can be an exponential number of constraints (2.2). (Figure 2.7 shows that this can really happen. The potentially exponential number of maximal cliques in arrow graphs distinguishes them from interval graphs, which have only a linear number of maximal cliques.)

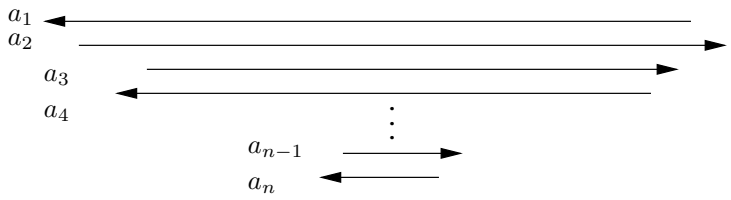


Figure 2.7: Example of an arrow graph with an exponential number of maximum cliques. For every choice of arrows from a compatible pair (a_{2i-1}, a_{2i}) we get a clique of size $n/2$, which is maximum. The arrow graph can arise from a 1D-JBS instance with two base stations in the middle and $n/2$ users on either side

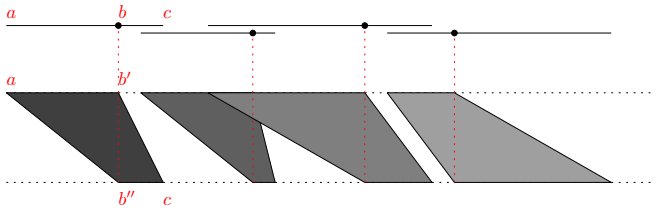


Figure 2.8: Example for interference segments

Fortunately, we can still solve such an LP in polynomial time with the ellipsoid method of Khachiyan [11] applied in a setting similar to [10]. This method only requires a separation oracle that provides us for any values of l_i, r_i with a violated constraint, if one exists. It is easy to check for a violation of constraints (2.3) and (2.4). For constraints (2.2), we need to check if for given values of l_i, r_i the maximum weighted clique in $G(A)$ is smaller than k. By Theorem 2.6 this can be done in time $\mathcal{O}(n \log n)$. Summarizing, we get the following theorem:

Theorem 2.10. *There is a polynomial-time 2-approximation algorithm for the 1D-JBS problem.*

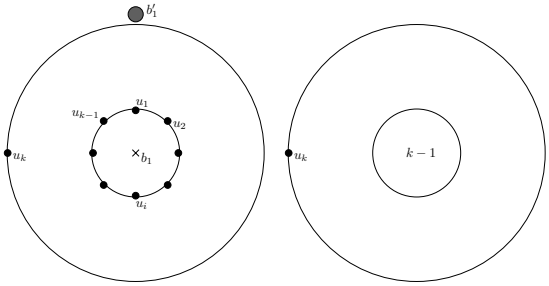
2.7 Different Interference Models

Until now we have analyzed the discrete interference model where the interference region has no effect beyond the targeted user. One step towards a more realistic model is to consider the interference region, produced by a base station sending a signal to a user, to span also beyond the targeted user. For the 1-dimensional case this can be modeled by using *interference segments* with the user somewhere between the endpoints of this segment (the small black circles on the segments in Figure 2.8) and the base station in the middle of the segment. The conflict graph of such interference segments is another special case of trapezoid graphs. For an example see Figure 2.8.

To see why the transformation to trapezoid graphs is correct, one should consider a segment with a user between its endpoints as two arrows pointing to the user from the left and the right, and then the triangle transformation (from Section 2.3) results in the trapezoid representation presented above. Hence for this generalization of the 1D-JBS problem we have a 2-approximation algorithm using the same technique as in Section 2.6.

3 General Case in the Plane—2D-JBS

Here we analyze the two-dimensional version (2D-JBS) of the base station scheduling problem. The decision variant k-2D-JBS of the problem asks for a given k and an instance of 2D-JBS whether the users can be served in at most k rounds. We show that k-2D-JBS is *NP*-complete for any $k \geq 3$. For the case $k = 1$, we give a polynomial algorithm. Then we present a constant-factor approximation algorithm for a constrained version of 2D-JBS. At the end of the section, we show logarithmic lower bounds on the approximation ratio of several natural greedy approaches.



(a) Realization with k users and one base station

(b) Simplified drawing

Figure 3.1: Example of a vertex connected to a $k-1$-clique

3.1 *NP*-Completeness of the k-2D-JBS Problem

In this section we show that the decision variant of the joint base station scheduling problem in the plane (k-2D-JBS) is *NP*-complete. We provide a reduction from the general k-colorability problem. Our reduction follows the methodology presented in [9] for unit disk k-colorability. We present the realization of the auxiliary graphs using 2D-JBS instances and show their correctness.

We recall the definition of the graph k-colorability problem presented in [8].

Graph k-Colorability: Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$. Is G k-colorable, i.e., does there exist a function $f : V \to \{1, 2, \ldots, k\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

Given any graph G, it is possible to construct in polynomial time a corresponding 2D-JBS instance that can be scheduled in k rounds if and only if G is k-colorable. We use an embedding of G into the plane which allows us to replace the edges of G with suitable base station chains with several users in a systematic fashion such that k-colorability is preserved. The *output vertices* of the auxiliary graphs represent the vertices of the original graph G.

In the following we recall the auxiliary graph structures from [9] used for the embedding and we show their realization using JBS instances. In the resulting JBS instances we make frequent use of cliques joined to single vertices (see Figure 3.1(a)). A clique of size $k-1$ is realized by a disk with $k-1$ users on the perimeter and one base station b_1 in the middle of the disk. We call this disk the *inner disk* of base station b_1 and the users on its perimeter the *inner users*. A vertex u_k connected to this clique is represented by a user on a bigger disk. We call this disk the *outer disk* of base station b_1 and the user(s) on its perimeter the *outer user(s)*. To force this base station b_1 to serve users only inside the disk $d(b_1, u_k)$ we place an *auxiliary* base station b'_1 with k users around it such that the distance from b_1 to any outer user of any other base station will be greater than the distance from b_1 to any user of b'_1 (see Figure 3.2). To simplify the drawings in what follows for the JBS instances we omit the auxiliary base stations and the users around them. Furthermore we draw the cliques as disks labeled with their size (see Figure 3.1(b)). In the computations the factor 2 for the number of base stations and number of users comes from counting also the auxiliary base stations and their users. We present now the definitions of the auxiliary graph structures from [9] and their realization by JBS instances together with their properties.

Definition 3.1. A *k-wire* of length l, denoted by W_k^l, consists of $l+1$ vertices $W_{v_0}, W_{v_1}, \ldots, W_{v_l}$ and l $(k-1)$-cliques $WC_1, WC_2, \ldots, WC_l$ such that for each $1 \leq i \leq l$ all vertices of the clique WC_i are connected to both $W_{v_{i-1}}$ and W_{v_i}. The vertices W_{v_0} and W_{v_l} are the output vertices of the k-wire (see Figure 3.3(a)).

The realization of the k-wire by a JBS instance is given in Figure 3.4(a). In this and the following JBS instances we have to show that the number of users remains polynomial as a function of the number of vertices of the original graph G and the transformation from G can

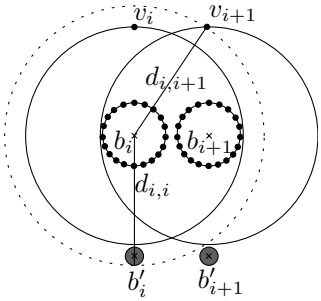
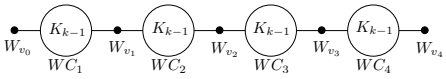
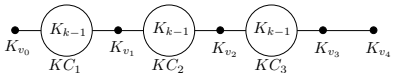


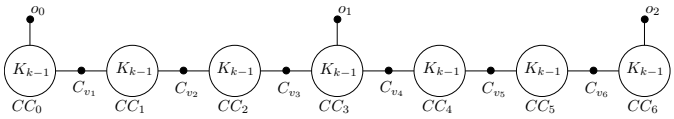
Figure 3.2: Forcing base stations to serve users only inside their outer disk



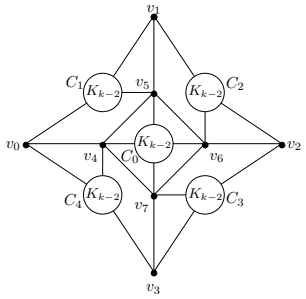
(a) The k-wire W_k^4



(b) The k-chain K_k^3

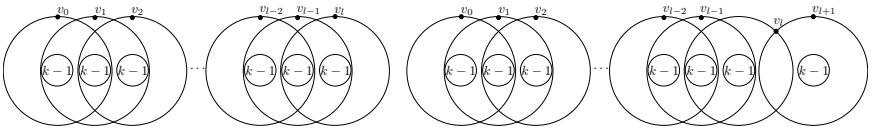


(c) The k-clone C_k^3 of size 3



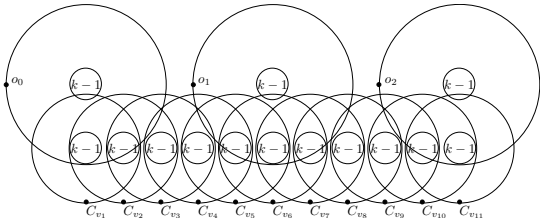
(d) The k-crossing H_k

Figure 3.3: Auxiliary graph structures



(a) The k-wire W_k^l realized by a JBS instance

(b) The k-chain K_k^l realized by a JBS instance



(c) The k-clone C_k^5 of size 3 realized by a JBS instance

Figure 3.4: Realization of the auxiliary graph structures by JBS instances

be done in polynomial time. We also have to show that the assignment of the users to base stations is uniquely defined if we want to preserve k-colorability.

Observation 3.2. *A JBS instance for a W_k^l has the following properties:*

1. *It has $m = 2(l+1)$ base stations and $n = 2k \cdot (l+1)$ users.*
2. *It can be scheduled in k rounds, but not in $k-1$.*
3. *Every k-scheduling assigns to every base station only its inner and outer users.*
4. *Every k-scheduling assigns the two output vertices v_0 and v_l to the same round.*

Proof. It is easy to observe that a W_k^l cannot be scheduled in less than k rounds since the lower bound on the number of necessary rounds is $\lceil \frac{n}{m} \rceil = k$. We also know that there is a valid k-schedule for a W_k^l since the schedule that assigns to every base station its inner and outer users needs exactly k-rounds. Since the number of users n is $k \cdot m$ we know that all valid k-schedules have to use all base stations in every round.

To prove that the only valid k-scheduling assigns to every base station only its inner and outer users we need the auxiliary base stations and the k users around them. First we prove that every base station (including the auxiliary ones) must serve its own inner users. Suppose that some base station b_j serves an inner user of base station b_i ($i \neq j$) in round l. In round l base station b_i cannot serve any other inner user since they are all at the same distance around b_i and hence only one of them can be served in a round. Base station b_i cannot serve any user that is farther away than its inner users in round l because the interference disk would block the inner user that is served by base station b_j. Hence base station b_i has to stay idle in round l which contradicts the validity of this schedule as a k-scheduling. We still have to prove that no base station serves the outer users of other base stations. Suppose that base station b_i serves the outer user of another base station, w.l.o.g. let this other base station be b_{i+1}. The auxiliary base station b_i' is placed in such a way that the distance $d_{i,i}$ is less than the distance $d_{i,i+1}$ (see Figure 3.2) and thus the interference disk $d(b_i, v_{i+1})$ would block the auxiliary base station b_i' from serving its inner users. This contradicts the fact that in a valid k-scheduling every base station is active in every round.

The last property holds since the interference that the outer users ($v_0, \ldots, v_l$) produce forces every valid k-schedule to schedule them in the same round. □

Definition 3.3. A *k-chain* of length l, denoted by K_k^l, consists of a W_k^l together with an additional vertex connected with one of the output vertices. This new vertex and the vertex at the other end of the original k-wire are the output vertices of the k-chain (see Figure 3.3(b)).

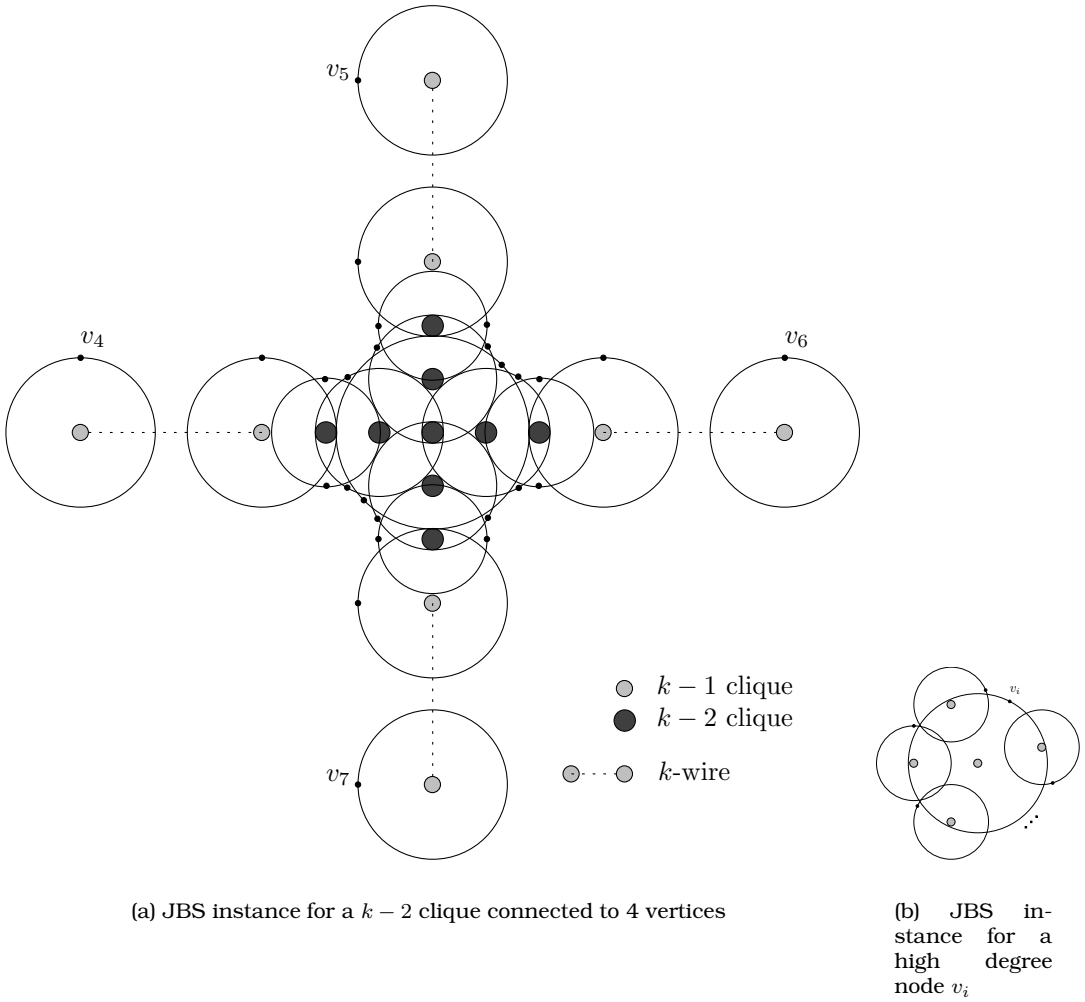
The realization of the k-chain using a JBS instance is given in Figure 3.4(b).

Observation 3.4. *A JBS instance for a K_k^l has the following properties:*

1. *It has $m = 2(l+2)$ base stations and $n = 2k \cdot (l+2)$ users.*
2. *It can be scheduled in k rounds, but not in $k-1$.*
3. *Every k-scheduling assigns to every base station only its inner and outer users.*
4. *Every k-scheduling assigns the two output vertices v_0 and v_{l+1} to different rounds.*

Proof. By the same argument as for the W_k^l one can prove the second property. From the properties of a k-wire we know that every valid k-schedule schedules the outer users $v_0 \ldots v_l$ in the same round. User v_l cannot be served by the right-most base station because then the base station to the left of it would have to stay idle in that round. Thus a valid k-schedule will assign to every base station only its inner and outer users. The user v_l is in the interference region of the user v_{l+1}, hence it forces every k-schedule to schedule them in different rounds. □

Definition 3.5. A *k-clone* of size $s \geq 2$ and length l, denoted by C_k^l, consists of the $l(s-1)$ vertices $C_{v_1}, C_{v_2}, \ldots, C_{v_{l(s-1)}}$, the $l(s-1)+1$ $(k-1)$-cliques $CC_0, CC_1, \ldots, CC_{l(s-1)}$ and s output vertices $o_0, o_1, \ldots, o_{s-1}$. Each C_{v_i}, $i = 1, \ldots, l(s-1)$ is connected to all vertices of CC_{i-1} and CC_i. Furthermore, each output vertex o_i, $i = 0, \ldots, s-1$ is connected to all vertices of $CC_{l \cdot i}$ (see Figure 3.3(c)).



(a) JBS instance for a $k-2$ clique connected to 4 vertices

(b) JBS instance for a high degree node v_i

Figure 3.5: Helper gadgets for realizing the k-crossing with JBS instances

The realization of the k-clone C_k^5 of size 3 using a JBS instance is given in Figure 3.4(c).

Observation 3.6. *A JBS instance for a C_k^l of size s has the following properties:*

1. *It has $m = 2(s + (s-1)l + 1)$ base stations and $n = 2k \cdot (s + (s-1)l + 1)$ users.*
2. *It can be scheduled in k rounds, but not in $k-1$.*
3. *Every k-scheduling assigns to every base station only its inner and outer users.*
4. *Every k-scheduling assigns the output vertices $o_0 \dots o_{s-1}$ to the same round.*

Proof. The proof is the same as for a k-wire. □

Definition 3.7. A *k-crossing*, denoted by H_k ($k \geq 3$), is represented by the graph in Figure 3.3(d). The vertices v_0, v_1, v_2, v_3 are the output vertices of the k-crossing.

We can realize the k-crossing using the following helper gadgets: for a $(k-2)$-clique connected to 4 vertices we can use the JBS instance from Figure 3.5(a). For a $(k-2)$-clique connected to 3 vertices we can use the same gadget having just 3 k-wires instead of 4. For a high degree node like v_4 or v_0 (from Figure 3.3(d)) we can use the JBS instance from Figure 3.5(b). The edges connecting the vertices can be realized with JBS instances for k-chains. The k-chains can be bent, they do not have to be vertical or horizontal.

Observation 3.8. *The JBS instance for a k-crossing H_k has the following properties:*

1. *It has m (constant) base stations and $n = k \cdot m$ users.*

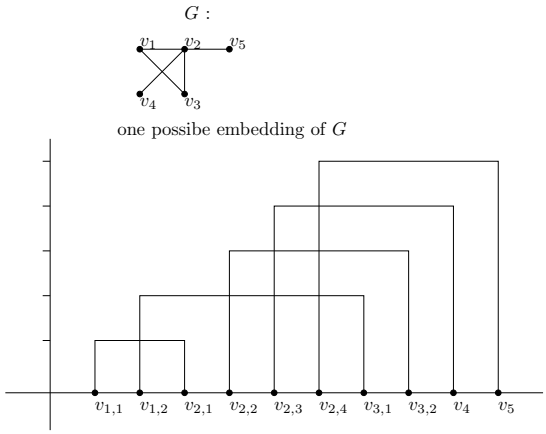


Figure 3.6: Example for the planar embedding

2. *It can be scheduled in k-rounds, but not in $k-1$.*
3. *Every k-scheduling assigns to every base station only its inner and outer users.*
4. *Every k-coloring (scheduling) f satisfies $f(v_0) = f(v_2)$ and $f(v_1) = f(v_3)$.*
5. *There exist k-colorings f_1 and f_2 which satisfy $f_1(v_0) = f_1(v_2) = f_1(v_1) = f_1(v_3)$ and $f_2(v_0) = f_2(v_2) \neq f_2(v_1) = f_2(v_3)$.*

Proof. The JBS instance for H_k is constructed in such a way that all base stations are fully loaded with k users. Hence it is obvious that it cannot be scheduled in less than k rounds. Similarly to the proof for the k-wire one can show that the only valid k-scheduling serves with every base station only its own inner and outer users. Since this JBS instance is a realization of the original k-crossing auxiliary graph we know that every k-coloring satisfies the last two properties. □

In what follows we give the main idea how to embed a general graph G into the plane using the auxiliary graph structures presented above. High degree vertices v_i are broken into an independent set of vertices $v_{i,1} \dots v_{i,d}$ ($d \geq 2$ is the degree of v_i). Each such vertex $v_{i,j}$ is then connected to one of the neighbors of v_i. The vertices are placed on a line such that the modified vertices coming from the same original vertex are placed next to one another. The original edges are replaced by horizontal and vertical lines connecting the corresponding vertices (or modified vertices). The properties that must hold in order to do the replacements without conflicting with each other are:

- All edges consist of horizontal and vertical line-segments.
- Between parallel line segments, vertices, and crossings certain minimal distances are preserved.
- These minimal distances ensure that at most two line segments cross in one point.
- For each given graph this embedding can be computed in polynomial time.

As an example in Figure 3.6 we show the embedding for an input graph G. The replacement vertices for a high degree node can be realized with a k-clone, the crossings with a k-crossing, the vertical and horizontal line segments with k-wires and k-chains. In [9] a systematic way is presented for doing these replacements using unit disk graphs and the same works here using instead JBS instances of the auxiliary graph structures.

Theorem 3.9. *The k-2D-JBS problem is NP-complete for any fixed $k \geq 3$.*

Proof. Let $G = (V, E)$ be any graph and $k \geq 3$. The construction of the corresponding embedding using JBS instances with its conflict graph $\hat{G} = (\hat{V}, \hat{E})$ can be done in polynomial time. All that remains to be shown is that

$$G \text{ is } k\text{-colorable} \Leftrightarrow \hat{G} \text{ is } k\text{-colorable.} \tag{3.1}$$

The proof is given in [9] and is based on the properties of the auxiliary graph structures. □

A direct consequence of the reduction used in proving Theorem 3.9 is the following corollary.

Corollary 3.10. *The coloring step of the k-2D-JBS problem is NP-complete for any fixed $k \geq 3$.*

Proof. In the reduction presented for the k-2D-JBS problem the selection of serving base stations for every user in the auxiliary graphs is uniquely determined by the construction. Hence the same reduction also works for just the coloring part of the k-2D-JBS problem. □

3.2 Base Station Assignment for One Round

The previous section showed that even if we have users assigned to the base stations as they are served in an optimal solution, the k-2D-JBS problem cannot be solved in polynomial time unless *P=NP*. Now we consider the complementary problem: knowing for every user the round in which she is served in a particular optimal solution, find an assignment of the users to the base stations such that a valid optimal schedule is obtained. We will see that this problem is solvable in polynomial time, which actually shows that, in some sense, the assignment problem is easier than the coloring one.

Knowing for every user the round in which she is served, we can consider every round as an independent problem by taking into account only the users scheduled in the corresponding round. We study therefore the problem of deciding whether we can serve all the users in one round.

We start with a simple observation that is valid also for the general 2D-JBS problem. Consider an *empty* disk $d = d(b, u)$ in the given setting of users U and base stations B, i.e. a disk containing only user u. We claim that every optimal solution can use d to serve u without changing the disks selected for other users. To see this, imagine u is served by some other base station b' in an optimal solution. Then b has to be idle in this round, because u is the closest user of b and therefore serving anybody else would block u. Moreover, d does not contain any other user, therefore we can serve u with b instead of b' in the same round without blocking anybody else.

We can adapt this idea to our setting. Suppose we can serve all the users in one round. Observe that every optimal solution is a set of empty disks. Consider the set of all possible empty disks formed by B and U (for every base station there is at most one empty disk, determined by the closest user to the base station[3]). For every user $u \in U$ there must be at least one empty disk serving u (e.g., the disk from an optimal solution). We know that we can use any of these disks in any optimal solution, therefore we can pick any empty disk into our optimal solution. So our algorithm simply computes all empty disks and then selects for each user u an arbitrary empty disk serving u; if some user u does not have an empty disk, the users cannot be served in one round.

The algorithm can clearly be implemented to run in polynomial time. Using standard techniques from computational geometry [5], e.g., computing in $\mathcal{O}(n \log n)$ time a Voronoi diagram for the user points and then a point location data structure so that the closest user of a base station can be determined in $O(\log n)$ time, we obtain a running-time of $\mathcal{O}((m + n) \log n)$.

Lemma 3.11. *The problem of deciding whether all users in a given 2D-JBS instance can be scheduled in one round can be solved in time $\mathcal{O}((n + m) \log n)$.*

Corollary 3.12. *Given the sets $U_1, \ldots, U_r$ of users scheduled in rounds $1, \ldots, r$ in an optimal solution, the problem of assigning base stations to the users such that we obtain a valid schedule of users U_i in round i can be solved in polynomial time.*

3.3 Approximation Algorithms

Looking for algorithms for the general 2D-JBS problem, we first analyze an approximation algorithm for a restricted setting where base stations have a limited transmission range and are not too close to each other. Then we discuss several greedy approaches and give lower bounds on their approximation ratios. The linear programming approach (as for 1D-JBS) cannot be applied directly: The conflict graph of a set of interference disks is not perfect, as is shown in Figure 3.7.

[3]There is no empty disk at b if there are 2 or more closest users at the same distance from b.

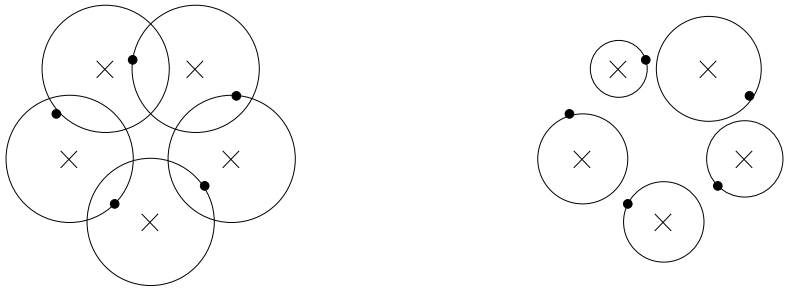


Figure 3.7: A cycle of length 5 in the conflict graph of interference disks (left). Note that it is not clear, however, whether an optimal solution to the selection problem will ever yield such a conflict graph; a different selection for this instance yields a conflict graph with five isolated vertices (right)

3.3.1 Bounded Geometric Constraints

We consider instances where the base stations are at least a distance Δ from each other and have limited power to serve a user, i.e., every base station can serve only users that are at most $R_{\max}$ away from it. We also assume that for every user there is at least one base station that can reach the user. We present a simple algorithm achieving an approximation ratio depending only on the parameters Δ and $R_{\max}$.

Consider the following greedy approach: For round $1, 2, \ldots$, the algorithm repeatedly picks an arbitrary user base-station pair (u, b), where u is an unserved user, such that the transmission from b to u can be added to the current round without creating a conflict. If no such user base-station pair exists, the next round starts. The algorithm terminates when all users have been served.

We can analyze the approximation ratio achieved by this greedy algorithm as follows. Assume that the algorithm schedules the users in k rounds. Let u be a user served in round k, and let b be the base station serving u. Since u was not served in rounds $1, 2, \ldots, k-1$, we know that in each of these rounds, at least one of the following is true:

- b serves another user $u' \neq u$.
- u is contained in an interference disk $d(b', u')$ for some user $u' \neq u$ that is served in that round.
- b cannot transmit to u because the disk $d(b, u)$ contains another user u' that is served in that round.

In each of these cases, we see that a user u' is served, and that the distance between u and u' is at most $2R_{\max}$ (since every interference disk has radius at most $R_{\max}$). Therefore, the disk with radius $2R_{\max}$ centered at u contains at least k users (including u). Let B' be the set of base stations that serve these k users in the optimal solution. The base stations in B' must be located in a disk with radius $3R_{\max}$ centered at u. As any two base stations are separated by a distance of Δ, we know that disks with radius $\Delta/2$ centered at base stations are interior-disjoint. Furthermore, the disks with radius $\Delta/2$ centered at the base stations in B' are all contained in a disk with radius $3R_{\max} + \Delta/2$ centered at u. Therefore, we have

$$|B'| \leq \frac{(3R_{\max} + \Delta/2)^2\pi}{(\Delta/2)^2\pi} = \frac{(6R_{\max} + \Delta)^2}{\Delta^2}.$$

Furthermore, we know that the optimal solution needs at least $k/|B'|$ rounds. This yields the following theorem.

Theorem 3.13. *There exists an approximation algorithm with approximation ratio $(\frac{6R_{\max}+\Delta}{\Delta})^2$ for 2D-JBS in the setting where any two base stations are at least Δ away from each other and every base station can serve only users within distance at most $R_{\max}$ from it.*

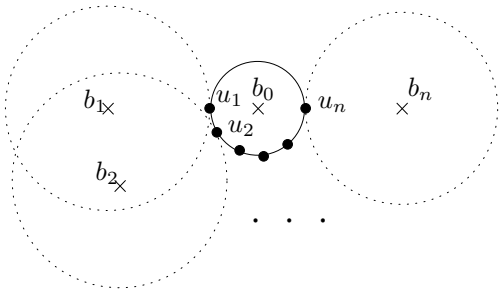


Figure 3.8: A greedy approach serves n users placed on a common interference disk in n time steps. An optimum algorithm can serve the users in one time step by assigning u_i to base station b_i, which lies on a halfline determined by b_0 and u_i

3.3.2 General 2D-JBS

We present lower bounds on the approximation ratios of three natural greedy approaches for the general 2D problem. The algorithms proceed round by round and use certain greedy rules to determine the user base-station pairs to be scheduled in the current round.

First, consider the greedy algorithm that serves as many users as possible in every round, i.e., in each round it chooses a maximum independent set in the conflict graph of all interference disks corresponding to user base-station pairs involving unserved users. We refer to this algorithm as the *maximum-independent-set* algorithm. The maximum independent set problem is *NP*-hard in general graphs, and we do not know its complexity for arrow graphs; nevertheless, we believe that it is interesting to determine the approximation ratio that can be achieved using this greedy approach, even if it is not clear whether the approach can actually be implemented in polynomial time.

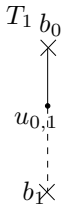
Furthermore, as in the previous section, we consider greedy algorithms that, in each round, repeatedly choose an interference disk of an unserved user that can be scheduled in the current round without creating a conflict. The algorithm that chooses among the interference disks of all unserved users a disk of smallest radius is the *smallest-disk-first* algorithm, and the algorithm choosing a disk containing the fewest other unserved users is the *fewest-users-in-disk* algorithm.

For the smallest-disk-first algorithm, there is a simple example (see Figure 3.8) showing that this approach has approximation ratio $\Omega(n)$. All the user points in this example, however, lie on the perimeter of a common interference disk. As such a configuration appears to be a rare case in practice, this leads us to consider instances of 2D-JBS in general position, defined as follows.

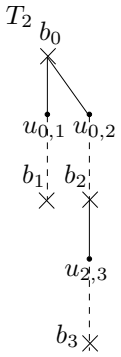
Definition 3.14. We say that sets of points $(U, B) \subset \mathbb{R}^2 \times \mathbb{R}^2$ are in *general position* if no two points from U lie on a circle centered at some point in B.

For points in general position we can show a lower bound of $\Omega(\log n)$ on the approximation ratio of all three considered greedy algorithms. Our construction of 2D-JBS instances leading to this lower bound is sketched in Figure 3.9. Conceptually, the arrangement of users and base stations can be viewed as a tree, where the edges of the tree are indicated by solid and dashed lines in the figure. The dashed lines represent a distance Δ, while the solid lines represent much shorter distances $\varepsilon_i \ll \Delta$. The structure of the tree is such that, after contracting the dashed edges, one obtains a binomial tree. The base stations in the figure are labelled by $b_0, b_1, \ldots$ in the order of a depth-first search traversal of the tree. Users are vertices of degree 2 and are adjacent to two base stations; our convention is to label a user adjacent to b_i and b_j with $u_{i,j}$.

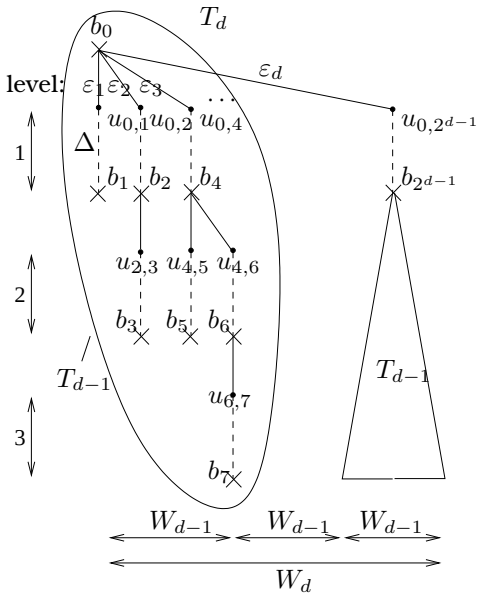
The base station at the root of the tree is b_0. The aim of the construction is to have the greedy algorithms use b_0 to serve the d users $u_{0,1}, u_{0,2}, u_{0,4}, \ldots, u_{0,2^i}, \ldots, u_{0,2^{d-1}}$, which are the children of b_0, in d consecutive rounds, whereas they can be served in a single round by an optimum algorithm by using the d base stations $b_1, b_2, b_4, \ldots, b_{2^i}, \ldots, b_{2^{d-1}}$. We show how to construct a tree T_d with this property for every value of d. In the following, the term "greedy algorithm" should be taken to refer to the smallest-disk-first algorithm. Afterwards we will discuss how the two other greedy algorithms behave on the constructed instances.



(a) Base station b_0 serves the user $u_{0,1}$ and blocks b_1



(b) b_0 blocks b_1 and b_2 blocks b_3 in the first round. In the second round, b_0 serves $u_{0,2}$ and blocks b_2



(c) Recursive construction of instances where greedy algorithms use b_0 to serve the d users directly below b_0 in d rounds

Figure 3.9: Outline of the construction of instances that greedy algorithms cannot handle efficiently

For $d = 1$, we simply put the user $u_{0,1}$ between base stations b_0 and b_1 at a position closer to b_0, see Figure 3.9(a). For $d = 2$, we have to ensure that user $u_{0,2}$ is not served by b_2 in the first round. Therefore we occupy b_2 in the first round by another user $u_{2,3}$, which will be selected by the greedy algorithm (see Figure 3.9(b)).

Similarly, for general d, we construct the tree T_d from two trees T_{d-1}, see Figure 3.9(c). We want to ensure that the user $u_{0,2^{d-1}}$ is not served in the first $d-1$ rounds; hence, we make base station $b_{2^{d-1}}$ (which could serve $u_{0,2^{d-1}}$ from below) busy for $d-1$ rounds by creating a copy of T_{d-1} rooted at $b_{2^{d-1}}$. Equivalently, the tree T_d can be viewed as putting trees $T_0, T_1, T_2, \ldots, T_{d-1}$ below the users $u_{0,1}, \ldots, u_{0,2^{d-1}}$. Note also that every base station in the tree T_d is the root of a tree T_k for some k. With this construction, we obtain a tree with levels of users and base stations, where level i consists of the users and base stations whose distance to the root (in terms of the number of edges) is of the form $2i-1$ or $2i$. All users in a level have the same y-coordinate, and the same holds for the base stations in a level. We set $\varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_d$ and Δ in such a way that $\varepsilon_d \ll \Delta$ and $W_d \ll \varepsilon_1$, where W_d is the width of the tree T_d (see Figure 3.9). When constructing a tree T_d from two trees T_{d-1}, we always leave a free space of width W_{d-1} between these two trees. We adjust Δ such that serving a user $u_{i,j}$ from level ℓ by base station b_j blocks all the users on level $\ell+1$. Also, we adjust W_d such that a disk centered at any user u from level i with radius ε_1 contains all other users from level i. Figure 3.10 depicts the desired property. To ensure that the points are in general position, we can adjust the x-coordinates of all users by a small perturbation that is much smaller than any of the values ε_i.

The tree is constructed in such a way that the greedy algorithm will pick the disks formed by base stations and users at distance ε_i in the i-th round, $1 \le i \le d$. Let G_i denote the set of these disks chosen in the i-th round. An optimal algorithm can serve all users on some level ℓ in one round by using the base stations from below, at the expense of blocking all users on level $\ell+1$. Thus, it can serve all users in two rounds by serving the odd levels in one round and the even levels in the other. Consequently, the approximation ratio of the greedy algorithm is $\Omega(d)$. The number n_d of users in the tree T_d can be calculated by solving the simple recursion

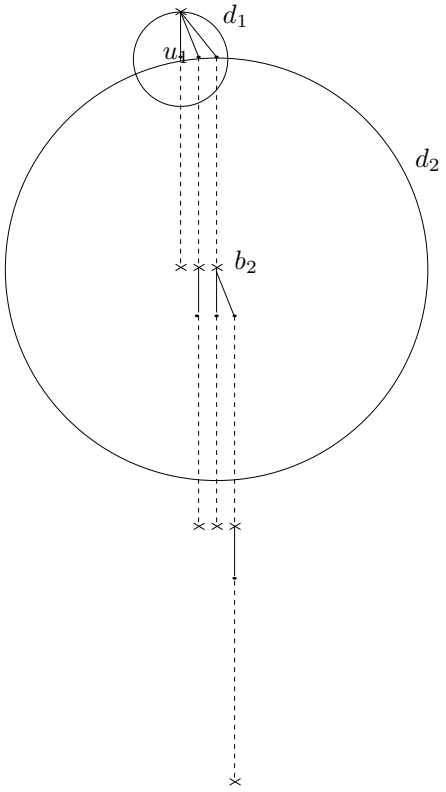


Figure 3.10: Possible realization of the "bad" instance for the greedy algorithms. The disk d_1 centered at u_1 contains all other users that are on the same level as u_1 is. The disk d_2 centered at b_2 blocks all users on the level below b_2

$n_d = 2n_{d-1} + 1$ (which follows from the recursive construction of the tree), where $n_1 = 1$. This gives $n = n_d = 2^d - 1$ users and thus $d = \log(n+1)$.

As every disk from G_i does not contain any unserved user, G_i is also a possible choice for the fewest-users-in-disk greedy algorithm. Thus we have shown the following.

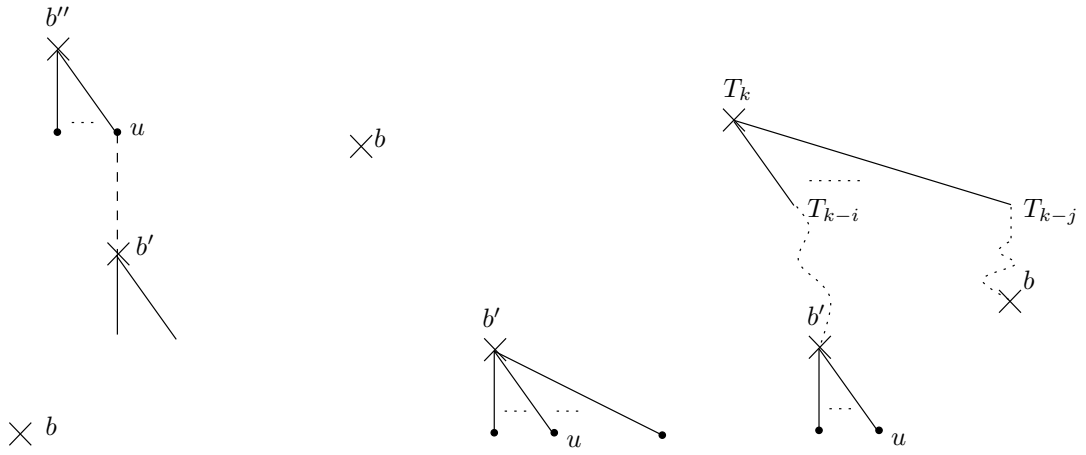
Lemma 3.15. *There are instances (U, B) of 2D-JBS in general position for which the smallest-disk-first greedy algorithm and the fewest-users-in-disk greedy algorithm have approximation ratio $\Omega(\log n)$, where $n = |U|$.*

Furthermore, we can show that G_i is a maximum independent set among the interference disks of all unserved users after the first $i-1$ rounds of the algorithm, implying that the greedy algorithm maximizing the number of served users in each round can produce the same solution as the two other greedy algorithms on this instance.

To show this, we consider an arbitrary maximum independent set M_i among the interference disks of all unserved users (after serving $G_1, \ldots, G_{i-1}$ in previous rounds) and show $|M_i| = |G_i|$. The inequality $|M_i| \geq |G_i|$ is obvious. To show the second inequality, we proceed as follows. Define an *active* base station as a base station with unserved neighboring user(s) below the base station. A base station such that all neighboring users below it have been served is called *passive*. We present several transformations on M_i (exchanging one interference disk by another) that preserve the size of M_i and the independence of the disks and lead to a new maximum independent set where only active base stations are used to serve users. This shows the desired inequality $|M_i| \leq |G_i|$, since in G_i all active base stations serve a user.

First, we transform M_i so that base stations serve only neighboring users. Suppose there is user u on level i that is served by a non-neighboring base station b in M_i. We perform a case study depending on the position of b.

1. Suppose b is below u (i.e., from level $i, i+1, \ldots$). Consider the bottom neighbor b' of u (from level i, cf. Figure 3.11(a)). We claim that b' is idle (i.e., b' does not serve any user); b' cannot serve any user above b', because this would block u (u is the closest user above b'); b' cannot serve any of its bottom neighbors (from level $i+1$) because all the users from



(a) Base station b serves user u from below. If base station b' is passive, then the upper neighbor b'' of u must be idle.

(b) Base station b serves user u. The base station is to the left and above of u. Upper neighboring base station b' of u has to be idle.

(c) Base station b is to the right and above of u.

Figure 3.11: Transformations for the maximum-independent-set algorithm

level $i+1$ are blocked by the disk $d(b,u)$; b' cannot serve any of the users from a level greater than $i+1$ because this would block u. Therefore we can serve u by b' (no new interference is created because the disk $d(b',u)$ blocks only the users on level $i+1$, which were blocked also by the disk $d(b,u)$).

2. Suppose b is above u (i.e., from level $i-1, i-2, \ldots$) and to the left of u (see Figure 3.11(b)). Consider the upper neighbor b' of u (b' is from level $i-1$; note also that b must be to the left of b', by construction of the instance); b' cannot serve any user above it, because that would block u; b' cannot serve any user that is to the right of u and below b' for the same reason; b' cannot serve any user from level greater than i for the same reason; b' cannot serve any neighboring user to the left of u because these users are blocked by the disk $d(b,u)$; b' cannot serve any user from level i to the left of its leftmost neighbor because then it would block u (as the horizontal distance is at least the width of the tree rooted at b'). Therefore, b' is idle and can serve u instead of b without introducing any new interference.

3. Suppose b is above u (i.e., from level $i-1, i-2, \ldots$) and to the right of u (see Figure 3.11(c)). Suppose b is the rightmost base station with the property of serving a user and being to the right and above of her. Consider the smallest subtree T_k containing both u and b. The children subtrees of T_k are of the form $T_1, T_2, \ldots, T_{k-1}$. Let T_{k-i} be the child subtree of T_k containing u and let T_{k-j} be the child subtree of T_k containing b. Note that $i > j$, therefore the horizontal distance from b to u is at least W_{k-j}. Thus, b also blocks all users on level i in the tree T_{k-j}, in particular, user u', which is the user in T_{k-j} whose position in T_{k-j} is identical to the position of u in T_{k-i} (note that, because the root of T_{k-j} has more children than that of T_{k-i}, there must be such a user u', and u' hasn't been served in previous rounds). We will show that u' is only in the interference disk $d := d(b,u)$, and therefore we can use b to serve u' instead of u (the interference disk gets smaller). Now if b is still to the right of u' we are left with a smaller tree T_{k-j} containing both b and u' and we can apply the same transformation rule 3 again until b is to the left of u', when we can apply the transformation 2.

   It remains to show that u' is only in the interference disk d. Suppose for a contradiction that u' is also in another disk $d'' = d(b'', u'')$. Consider the case where b'' is above u'. If user u'' is above b'' then u'' is a neigbor of b'' (transformation 1 has been applied) and therefore if d'' blocks u' then it blocks also the whole level i, u included. So u'' must be below b''. Then u'' has to be from level i (u'' being on level greater than i blocks the whole level i). If

u'' is a child of b'' then also u' is, and therefore u'' is blocked by d. Therefore b'' is not a neighbor of u'' and b'' is to the right of u'' (we have applied transformation 2 already), and therefore d'' does not block any user to the left of u''. Thus u'' is to the left of u' and to the right of u. But then u'' is blocked by the disk d.

Consider the case where b'' is below u'. If u'' is above b'', we know that u'' is the upper neighbor of b'' (we have applied transformation 1 already), and therefore u'' is the only user above b'' that is blocked by d'', a contradiction to the assumption that d'' contains u'. If u'' is below b'', for d'' to reach u' the radius has to be at least ε_1 more than the distance from b'' to u'. Therefore, d'' blocks the whole level i, u included.

Now we have a maximum independent set M_i where base stations serve only neighboring users. If base station b' serves a neighbor below, then clearly b' is active. If b' is passive and serves its upper neighbor u, then consider b'' (cf. Figure 3.11(a)), the upper neighbor of u, which is active. We claim that b'' is idle. Because b' is passive and u is its upper neighbor, u has to be the first unserved user (in left-to-right order) among the children of b''. Therefore, if b'' were to serve any other user, it would block u. Thus, we can use b'' to serve u instead of b'. Finally, we have obtained a maximum independent set where only active base stations serve users.

Theorem 3.16. *There are instances (U, B) of 2D-JBS in general position for which the maximum-independent-set algorithm, the smallest-disk-first greedy algorithm, and the fewest-users-in-disk algorithm have approximation ratio $\Omega(\log n)$, where $n = |U|$.*

We note that the algorithm maximizing the number of served users in every round achieves approximation ratio $\mathcal{O}(\log n)$, as can be shown by applying the standard analysis of the greedy set covering algorithm.

4 Conclusion and Open Problems

In this paper we study the 1D- and 2D-JBS problems that arise in the context of coordinated scheduling in packet data systems. These problems can be split into a selection and a coloring problem. In the one-dimensional case, we have shown that the coloring problem leads to the class of arrow graphs, for which we have discussed its relation to other graph classes and algorithms. For the selection problem we propose an approach based on LP relaxation and rounding. For the 2D-problem, we have shown its NP-completeness. Several problems remain unsolved. In particular, it is open whether the 1D-JBS problem is *NP*-complete. For 2D-JBS it would be interesting to design approximation algorithms whose approximation ratio does not depend on the ratio $\frac{R_{\max}}{\Delta}$. Moreover, algorithmic results for more refined models would be interesting.

References

[1] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes: A survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[2] J. Chuzhoy and S. Naor. New hardness results for congestion minimization and machine scheduling. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing (STOC'04)*, pages 28–34, 2004.

[3] M. Cielibak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling jobs on a minimum number of machines. In *Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science*, pages 217–230. Kluwer, 2004.

[4] S. Das, H. Viswanathan, and G. Rittenhouse. Dynamic load balancing through coordinated scheduling in packet data systems. In *Proceedings of Infocom'03*, 2003.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications*. Springer, 2000.

[6] T. Erlebach and F. C. R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *Algorithmica*, 46:27–53, 2001.

[7] S. Felsner, R. Müller, and L. Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74:13–32, 1997.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.

[9] A. Gräf, M. Stumpf, and G. Weißenfels. On coloring unit disk graphs. *Algorithmica*, 20(3):277–293, March 1998.

[10] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[11] L. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[12] F. C. R. Spieksma. On the approximability of an interval scheduling problem. *Journal of Scheduling*, 2:215–227, 1999.

[13] J. P. Spinrad. *Efficient Graph Representations*, volume 19 of *Field Institute Monographs*. AMS, 2003.

[14] D. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2001.