Ordering with precedence constraints and budget minimization

Jeff Kinne¹, Ján Maňuch², Akbar Rafiey³, and Arash Rafiey¹,³

¹ Indiana State University, IN, USA,
² University of British Columbia, BC, Canada
³ Simon Fraser University, BC, Canada

Abstract. We introduce a variation of the scheduling with precedence constraints problem that has applications to molecular folding and production management. We are given a bipartite graph $H = (B, S)$. Vertices in $B$ are thought of as goods or services that must be bought to produce items in $S$ that are to be sold. An edge from $j \in S$ to $i \in B$ indicates that the production of $j$ requires the purchase of $i$. Each vertex in $B$ has a cost, and each vertex in $S$ results in some gain. The goal is to obtain an ordering of $B \cup S$ that respects the precedence constraints and maximizes the minimal net profit encountered as the vertices are processed. We call this optimal value the budget or capital investment required for the bipartite graph, and refer to our problem as the bipartite graph ordering problem.

The problem is equivalent to a version of an NP-complete molecular folding problem that has been studied recently [14]. Work on the molecular folding problem has focused on heuristic algorithms and exponential-time exact algorithms for the un-weighted problem where costs are ±1 and when restricted to graphs arising from RNA folding.

The present work seeks exact algorithms for solving the bipartite ordering problem. We demonstrate an algorithm that computes the optimal ordering in time $O^*(2^n)$ when $n$ is the number of vertices in the input bipartite graph. Our main result is a general strategy that can be used to find an optimal ordering in polynomial time for bipartite graphs that satisfy certain properties. We apply the technique to a variety of graph classes, obtaining polynomial-time solutions to the bipartite graph ordering problem for bipartite graphs that are convex, trivially perfect, co-bipartite graphs, and trees. One of our ultimate goals is to completely characterize the classes of graphs for which the problem can be solved exactly in polynomial time.

1 Motivation and Introduction

Job Scheduling with Precedence Constraints The setting of job scheduling with precedence constraints is a natural one that has been much studied (see, e.g., [6, 17]). A number of variations of the problem have been studied; we begin by stating one. The problem is formulated as a directed acyclic graph where the vertices are jobs and arcs between the vertices impose precedence constraints. Job $j$ must be executed after job $i$ is completed if there is an arc from $j$ to $i$. Each job $i$ has a weight $w_i$ and processing time $t_i$. A given ordering of executing the jobs results in a completion time $C_i$ for each job. Previous work has focused on minimizing the weighted completion time $\sum_{i=1}^{n} w_i C_i$. This can be done in the single-processor or multi-processor setting, and can be considered in settings where the precedence graph is from a restricted graph class. The general problem of finding an ordering that respects the precedence constraints and minimizes the weighted completion time is NP-complete. Both approximation algorithms and hardness of approximation results are known [2, 3, 17, 20].

Our Problem – Optimizing the Budget In the present work, we consider a different objective than previous works. In our setting, each job $j$ has a net profit (positive or negative) $p_j$. Our focus is on the budget required to realize a given ordering or schedule, and we disregard the processing time. We imagine that the jobs are divided between those with negative $p_i$, jobs $B$ that must be bought, and jobs with a non-negative $p_i$, jobs $S$ that are sold. $B$ could consist of raw inputs that must be...
purchased in bulk in order to produce goods $S$ that can be sold. A directed graph $H = (B, S)$ encodes the precedence constraints inherent in the production: an arc from $j \in S$ to $i \in B$ implies that item $i$ must be bought before item $j$ can be produced and sold. At each step $1 \leq r \leq n$ of the process, let $j_1, j_2, ..., j_r$ be the jobs processed thus far, and let $bg_r = \sum_{i=1}^{r} p_i$ be the total budget up to this point. Our goal is an ordering that respects the precedence constraints and keeps the minimal value of $bg_r$ as high as possible. One can view (the absolute value of) this optimal value as the capital investment required to realize the production schedule.

In this work we assume $H$ is a bipartite graph with all arcs from $S$ to $B$. This models the situation where each item to be produced and sold depends on certain inputs that must be purchased. We call this the problem of *ordering with precedence constraints and budget minimization on bipartite graphs* but refer to the problem as the *bipartite graph ordering problem*.

**Applications** The bipartite graph ordering problem is a natural variation of scheduling with precedence constraints problems. As described above the problem can be used to model the purchase of supplies and production of goods when purchasing in bulk. Another way to view the problem is that the items in $B$ are training sessions that employees must complete before employees (vertices in $S$) can begin to work.

We began studying the problem as a generalization of an optimization problem in molecular folding. The folding problem asks for the energy required for secondary RNA structures to be transformed from a given initial folding configuration $C_1$ into a given final folding configuration $C_2$ [9, 16, 19]. The bipartite graph ordering problem models this situation as follows: vertices in $B$ are folds that are to be removed from $C_1$, vertices in $S$ are folds that are to be added, and an edge from $j$ to $i$ indicates that fold $i$ must be removed before fold $j$ can be added. The price $p_i$ of a vertex is set according to the net energy that would result from allowing the given fold to occur, with folds that must be broken requiring a positive energy and folds that are to be added given a negative energy. The goal is to determine a sequence of transformations that respects these constraints and still keeps the net energy throughout at a minimum. Figure 1 shows how an instance of the RNA folding problem is transformed into the bipartite graph ordering problem.

**Previous Work** The molecular folding problem has been studied only in the setting of unit prices and most attention has been devoted to graph classes corresponding to typical folding patterns (in particular for so-called circle bipartite graphs). [14] shows that the molecular folding problem is NP-complete even when restricted to circle bipartite graphs; thus the bipartite graph ordering problem is NP-complete as well when restricted to circle bipartite graphs.

Previous work on the folding problem has focused on exact algorithms that take exponential time and on heuristic algorithms [8].

There has been considerable study of scheduling with precedence constraints, but to our knowledge there has not been any work by that community on the objective function we propose (budget minimization).

### 1.1 Our Results

We introduce the bipartite graph ordering problem, which is equivalent to a generalization of a molecular folding problem. We initiate the study of which graph classes admit polynomial-time exact solutions. We also give the first results for the weighted version of the problem; previous work on the molecular folding problem assumed unit costs for all folds.

---

4 Note that the molecular folding problem is a minimization problem, and can be made a maximization problem by negating the energies.

5 A graph $G$ is called a *circle graph* if the vertices are the chords of a circle and two vertices are adjacent if their chords intersect. The circle bipartite graphs can be represented as two sets $A, B$ where the vertices in $A$ are a set of non-crossing arcs on a real line and the vertices in $B$ are a set of non-crossing arcs from a real line; there is an edge between a vertex in $A$ and a vertex in $B$ if their arcs cross. The top graph in Figure 1 is a circle bipartite graph shown with this representation.
Ordering with precedence constraints and budget minimization

Fig. 1. The top graph is an instance of the RNA folding problem, with folds 1, 2, and 3 to be removed (bought), folds a, b, and c to be added (sold); an edge cannot be added until edges that cross it are removed. A budget of two is needed and an optimal ordering is 3, 2, b, 1, a, c.

**Exponential-time Exact Algorithm** We first give an exact algorithm for arbitrary bipartite graphs.

**Theorem 1.** Given a bipartite graph $H = (B, S)$, the bipartite graph ordering problem on $H$ can be solved in (a) time and space $O^*(2^n)$, and (b) time $O^*(4^n)$ and polynomial space, where $n = |B \cup S|$.

The previous best exact algorithm for the molecular folding problem on circle bipartite graphs has running time $n^{O(K)}$, where $K$ is the optimal budget [19].

We observe that $K$ can be $\Omega(n)$ when vertex prices are $\pm 1$ (and can be much larger when vertex prices can be arbitrary), as follows. Let $\mathcal{P}$ be a projective plane of order $p^2 + p + 1$ with $p$ prime. The projective plane of order $n = p^2 + p + 1$ consists of $n$ lines each consisting of precisely $p + 1$ points, and $n$ points which each are intersected by precisely $p + 1$ lines. We construct a bipartite graph with each vertex in $B$ corresponding to a line from the projective plane, each vertex in $S$ corresponding to a point from the projective plane, and a connection from $b \in B$ to $s \in S$ if the projective plane point corresponding to $s$ is contained in the line corresponding to $B$. Vertices in $B$ are given weight -1, and vertices in $S$ are given weight 1. Note that the degree of each vertex in $B$ is $p + 1$. One can observe that the neighbourhood of every set of $p + 1$ vertices in $S$ is at least $p^2 - \binom{p}{2}$. This implies that in order to be able to sell the first $p + 1$ vertices in $S$ the budget decreases by at least $p^2 - \binom{p}{2} + p$.

**Polynomial-time Cases** Our main result gives a general technique that can be used to solve the bipartite graph ordering problem for classes of bipartite graphs that satisfy certain properties. We apply the technique to a number of bipartite graph classes. These bipartite graph classes are briefly defined after the theorem statement and discussed further in Sections 6 and 7.

**Theorem 2.** Given a bipartite graph $H = (B, S)$, the bipartite graph ordering problem on $H$ can be solved in polynomial time if $H$ is one of the following: a convex bipartite graph, a trivially perfect bipartite graph, a co-bipartite graph or a tree.

The bipartite graphs we consider here have been considered for other types of optimization problems. In particular convex bipartite graphs (those for which there exists an ordering of the vertices in $B$ where the neighborhood of each vertex in $S$ is a set of consecutive vertices) are of interest in biology [1, 13] and also energy production applications where resources (in our case bought vertices) can be assigned (bought) and used (sold) within a number of successive time steps [12]. There are several recognition algorithms for convex bipartite graphs [11, 15]. A bipartite graph is called trivially perfect if it is obtained from a union of two trivially perfect bipartite graphs $H_1, H_2$ or by joining every sold vertex in trivially perfect bipartite graph $H_1$ to every bought vertex in trivially perfect bipartite graph $H_2$. A single vertex is also a trivially perfect bipartite graph. These bipartite graphs have been considered in [5, 7, 17]. Co-bipartite graphs have a similar definition with a slightly different join operation. See Section 6 for the precise definitions.

For trivially perfect bipartite graphs and co-bipartite graphs, due to the recursive nature of the definition of these graphs it is natural to attempt a divide and conquer strategy. However, a simple approach of solving sub-problems and using these to build up to a solution of the whole problem...
fails because one may need to consider all possible orderings of combining the sub problems. The main difficulty to overcome in proving Theorem 2 is to determine which subgraph to process first, and we develop a general approach that applies to the graph classes mentioned. We frame our algorithm in a way amenable to extending the result to include other graph classes, with the hope of fully characterizing the graphs for which our problem is polynomially solvable. We observe in Section 8 that the algorithm applies to a class of graphs more general than convex graphs.

*Arbitrary Vertex Weights* Each of our results holds where the weights on vertices can be arbitrary (not only ±1 as considered by previous work on the molecular folding problem).

## 2 Some Simple Classes of bipartite graphs

In this section we state some simple facts about the bipartite graph ordering problem and give a simple self-contained proof that the problem can be solved for trees. We provide this section to assist the reader in developing an intuition for the problem.

**Bicliques** First we note that if $H$ is a biclique with $|B| = K$ then $bg(H)$ (the budget required to process $H$) is $K$.

As a next step, consider a disjoint union of bicliques $H_1, H_2, \ldots, H_m$ where each $H_i$ is a biclique between bought vertices $B_i$ and sold vertices $S_i$. Intuition suggests that we should first process those $H_i$ such that $|S_i| \geq |B_i|$. This is indeed correct and is formalized in Lemma 1 in Section 4 (the reader is encouraged to take this intuition for granted while initially reading the present section). After processing $H_i$ with $|S_i| \geq |B_i|$, which we call *positive* (formally defined in generality in Section 4), we are left with bicliques $H_i = (B_i, S_i)$ where $|B_i| > |S_i|$. Up to this point we may have built up some positive budget.

In processing the remaining $H_i$ the budget steadily goes down – because the $H_i$ are bicliques and disjoint, and the remaining sets are not positive. As we shall see momentarily, we should process those $H_i$ with largest $|S_i|$ first. Suppose on the contrary that $|S_i| > |S_j|$ but an optimal strategy $opt$ processes $H_j$ right before $H_i$. If $K$ is the budget before this step we first have that $K - |B_i| + |S_i| \geq |B_i|$ because otherwise there would not be sufficient budget after processing $H_j$ to process $H_i$. Since we assumed that $|S_i| > |S_j|$ we have $K - |B_i| + |S_i| \geq |B_j|$. Thus, we could first process $H_i$ and then $H_j$. We have thus given a method to compute an optimal strategy for a disjoint union of bicliques: first process positive sets, and then process bicliques in decreasing order of $|S_i|$.

**Paths and Cycles** We next consider a few even easier cases. Note that a simple path can be processed with a budget of at most 2, and a simple cycle can be processed with a budget of 2.

**Trees and Forests** Next we assume the input graph is a tree and the weights are $-1, 1$ (for vertices in $B$ and $S$, respectively). Let $H$ be a tree, or in general a forest. Note that any leaf has a single neighbor (or none, if it is an isolated vertex). We can thus immediately process any sold leaf $s$ by processing its parent in the tree and then processing $s$. This requires an initial budget of only 1. After repeating the process to process all sold leaves in $S$, we are left with a forest where all leaves are bought vertices in $B$. We can first remove from consideration any disconnected bought vertices in $B$ (these can, without loss of generality, be processed last). We are left with a forest $H'$.

We next take a sold vertex $s_1$ (which is not a leaf because all sold leaves in $S$ have already been processed) and process all of its neighbours. After processing $s_1$ we can process $s_1$ and return 1 unit to the budget. Note that because $H'$ is a forest, the neighbourhood of $s_1$ has intersection at most 1 with the neighbourhood of any other sold vertex in $S$. Because we have already processed all sold leaves from $H$, we know that only $s_1$ can be processed after processing its neighbours.

After processing $s_1$, we may be left with some sold leaves in $S$. If so, we deal with these as above. We note that if removing the neighbourhood of $s_1$ does create any sold leaves, then each of these has at least one bought vertex in $B$ that is its neighbour and is not the neighbour of any of the other sold leaves in $S$. When no sold leaves remain, we pick a sold vertex $s_2$ and deal with it as we did $s_1$. 

4 Jeff Kinne, Ján Maňuch, Akbar Rafiey, and Arash Rafiey
This process is repeated until all of $H'$ is processed. We note that after initially dealing with all sold leaves in $S$, we gain at most a single sold leaf at a time. That is, the budget initially increases as we process sold vertices and process their parents in the tree, and then the budget goes down progressively, only ever temporarily going up by a single unit each time a sold vertex is processed. Note that the budget initially increases, and then once it is decreasing only a single sold vertex is processed at a time. This implies that the budget required for our strategy is $|B| - |S| + 1$, the best possible budget for a graph with 1, -1 weights.

3 An Exponential-time Exact Algorithm

In this section we prove Theorem 1.

The authors in [4] show that any vertex ordering problem on graphs of a certain form can be solved in both (a) time and space $O^*(2^n)$, and (b) time $O^*(4^n)$ and polynomial space, where $n$ is the number of vertices in the graph and $O^*(f(n))$ is shorthand for $O(f(n) \cdot \text{poly}(n))$. We show that the ordering problem can be seen to have the form needed to apply this result.

A vertex ordering on graph $H = (B, S)$ is a bijection $\pi : B \cup S \rightarrow \{1, 2, \cdots, |B \cup S|\}$. Note that orderings we consider here respect the precedence constraints given by edges of bipartite graph $H$. For a vertex ordering $\pi$ and $v \in B \cup S$, we denote by $\pi_{\prec v}$ the set of vertices that appear before $v$ in the ordering. More precisely, $\pi_{\prec v} = \{ u \in B \cup S | \pi(u) < \pi(v) \}$.

Let $H(Q)$ be the set of all permutations of a set $Q$ and $f$ be a function that maps each couple consisting of a graph $H = (B, S)$ and a vertex set $Q \subseteq (B \cup S)$ to an integer as follows:

$$f(H, Q) = |Q \cap S| - |Q \cap B|.$$  

Note that the function $f$ is polynomially computable. Now, if we restrict the weights of vertices to be ±1 (vertices in $B$ have weight -1 and vertices in $S$ have weight 1) we can express the bipartite graph ordering problem as follows:

$$\text{bg}(H) = \min_{\pi \in H(B \cup S)} \max_{v \in (B \cup S)} f(H, \pi_{\prec v}).$$  

The right hand side of this equation is the form required to apply the result of [4], proving Theorem 1 for the case of ±1 weights. The result for arbitrary weights $p_x$, with $p_x$ negative for $x \in B$ and $p_y$ non-negative for $y \in S$, follows by modifying $f(H, Q)$ to be $\sum_{y \in Q \cap S} p_y - \sum_{z \in Q \cap B} p_z$.

4 Definitions and Concepts

In this section we define key terms and concepts that are relevant to algorithms that aim to solve the bipartite graph ordering problem on any type of bipartite graph. We use the graph in Figure 2 as an example to demonstrate each of our definitions. The reader is encouraged to consult the figure while reading this section. The bipartite graph ordering problem was defined in Section 1. Recall that for a given graph $H$, $\text{bg}(H)$ is the optimal budget required to process the graph.

Let $I$ be a set of vertices, $|I|$ refers to the cardinality of set $I$. When applying our arguments to weighted graphs, with vertex $x$ having price $p_x$, we let $|I|$ be defined as $\sum_{x \in I} |p_x|$. Each of our results holds for weighted graphs by letting $|I|$ refer to the weighted sum of vertices in $I$ for all definitions and arguments.

We use $K$ to denote the budget or capital available to process an input bipartite graph. As vertices are processed, we let $K$ denote the current amount of capital available for the rest of the graph.

**Definition 3 (N*(I))** Let $H = (B, S)$ be the input bipartite graph associated to the input problem. For a subset $I \subseteq B$ of bought vertices in $H$, let $N^*(I)$ be the set of all vertices in $S$ whose entire neighborhood lie in $I$.

**Definition 4 (prime I)** We say $I \subseteq B$ is prime if $N^*(I)$ is non-empty and for every proper subset $I' \neq I$, $N^*(I')$ is empty.
Note that the graph induced by prime set $I$ and $N^*(I)$ is a bipartite clique. For any strategy to process an input bipartite graph $H$, we look at the budget at each step of the algorithm. Suppose our initial budget is $K$. If we knew which subsets of $B$ are prime, then the first choice is which prime subset $I$ to process first – this would leave a budget of $K - |I| + |N^*(I)|$ to process the rest of the bipartite graph.

An algorithm will generally try to first process subsets $I$ that increase (or at least) decrease the budget. We call such subsets positive, and call $I$ negative if processing it would decrease the budget.

**Definition 5 (budget of $I$)** A budget of $I \subseteq B$ is the minimum capital $r$ needed to process $I \cup N^*(I)$, i.e. $bg(H[I, N^*(I)]) = r$. For simplicity we write $bg(I) = r$.

**Definition 6 (positive, negative $I$)** $I \subseteq B$ is called positive if $|I| \leq |N^*(I)|$ and it is negative if $|I| > |N^*(I)|$. For a given budget $K$, $I$ is called positive minimal (with respect to $K$) if it is positive, and $I$ has budget at most $K$, and every other positive subset of $I$ has budget more than $K$. In other words, $I$ is smallest among all the subsets of $I$ that is positive and has budget at most $K$.

Note that there can in general be more than one positive minimal set. Positive minimal sets are key in algorithms for computing the budget because these are precisely the sets that we would like to process first. In the graph of Figure 2, the positive set $I_1$ would be the first to be processed.

**Lemma 1.** Let $H = (B, S)$ be a bipartite graph that can be processed with budget at most $K$. Suppose $H$ contains a positive minimal set $I \subseteq B$ (with respect to budget $K$). Then there is a strategy for $H$ that has optimal budget and begins by processing $I$.

**Proof:** Suppose the optimal process $opt$ does not process $I$ all together and hence processes the sequence $L_1, L_2, I_2, L_2, I_1, L_1, I_1, L_1+1$ of disjoints subset of $B$ where $I = I_1 \cup I_2 \cup \ldots \cup I_t$ is a positive minimal set and $L_j \neq \emptyset$, $2 \leq j \leq t$. Note that according to $opt$ for all $I_1, 1 \leq i \leq t + 1$ we have $bg(L_i) \leq K - |S_2| + N^*(S_2)$ where $S_2 = \cup_{j=1}^{i-1} L_j \cup_{j=1}^{i-1} I_j$. Observe that since $I$ is positive minimal, $K - |S_2| + N^*(S_2) \leq K - |I \cup S_1| + N^*(I \cup S_1)$ where $S_1 = \cup_{j=1}^{i-1} L_j$. Therefore $bg(L_i)$ is at most $K - |I \cup S_1| + N^*(I \cup S_1)$ where $H_1 = H \setminus (I \cup S_1)$. Together with $bg(I) \leq K$, we conclude that, there is another optimal process that considers $I$ first and then $L_1, L_2, \ldots , L_t, L_{t+1}$ next.

Given a bipartite graph $H$, Lemma 1 suggests a basic strategy, if positive sets can be identified, is to identify a positive minimal subset $I$, process it, and repeat this process. When a given subset $I$ is processed, we would consider the remaining bipartite graph and again try to find a positive minimal subset to process, if one exists. Note that $H \setminus (I \cup N^*(I))$ may have positive sets even if $H$ does not. For example, in the graph of Figure 2, $H' = (J_2 \cup J_1 \cup J, D \cup E \cup F)$ has no positive set, but $J$ is positive in $H' \setminus J_1$. When a subset $I \subseteq B$ is processed we generally would like to process any sets that are positive in the remaining bipartite graph. That is, we would like to process $cl(I)$, defined as follows. For our purpose we order all the prime sets lexicographically, by assuming some ordering on the vertices of $B$.

**Definition 7 ($cl(I)$)** Given current budget $K$, let $cl_K(I) = I \cup_{i=1}^{r} I_i$ where $I_i \subseteq B$, $1 \leq i \leq r$ is the lexicographically first positive minimal subset in $H_i = H \setminus (\cup_{j=0}^{i-1} I_j | N^*(\cup_{j=0}^{i-1} I_j))$ such that $bg(I_{i+1}) \leq K - |I_i \cap B| + |H_i \cap S|$. When the initial budget $K$ is clear from context, we use $cl(I)$ rather than $cl_K(I)$.

Note that $cl(I)$ is well-defined. Here $r$ is the number of times the process of processing a positive minimal set can be repeated after processing $I$. Note that $cl(I)$ could be only $I$, in this case $r = 0$.

### 4.1 General Strategy

It may not always be the case that all positive sets can be identified in polynomial times. But, if positive sets can be identified, the following is a general strategy for processing an input bipartite graph $H$ and given budget $K$. 

A convex graph that we use as an example for the definitions related to our algorithm. Each bold line shows a complete connection, i.e. the induced subgraph by $I \cup L$ is a biclique. The numbers in the boxes are the number of vertices. The sets $J_1, J_2, J, I, I_1$ are the items $B$ to be bought, with each vertex having weight $-1$. The sets $D, E, F, L, P, O$ are the items $S$ to be sold, with each vertex having weight $1$. The prime sets (Definition 4) are: $\{J_1, J_2\}$, $J, I, I_1$. $I_1$ is the only positive set (Definition 6) with $N^*(I_1) = O$ containing 7 vertices, see Definition 3. We also have that $\ell(I) = J \cup J_1$ (with respect to any current budget $K$ at least 12 after removing $I_1$, $O$ see Definition 7) and Superset$(I) = J \cup J_1 \cup J_2$ (with respect to any current budget $K$ at least 12 after removing $I_1$, $O$ see Definition 9).

1. If there exist positive sets in $B$, process a positive minimal set $I$, setting $H = (B \setminus I, S \setminus N^*(I))$, update $K$ to be $K - |I| + |N^*(I)|$ and goto step 1.
2. If no positive sets exist, choose in some way the next prime set $I$ to process, set $H = (B \setminus I, S \setminus N^*(I))$, update $K$ to be $K - |I| + |N^*(I)|$ and goto step 1.

Note that each time a prime set $I$ is processed, we end up processing $\ell(I)$. Even if we can identify the prime and positive sets, it remains to determine in the second step the method for choosing the next prime to process. We address this issue and give the full algorithm and proof for Theorem 2 in the next section. Note that Lemma 1 implies that without loss of generality we can assume that when a prime set $I$ is processed the remainder of $\ell(I)$ is processed next.

**Corollary 1.** Let $H = (B, S)$ be a bipartite graph that can be processed with budget at most $K$ with an ordering that processes prime set $I$ first. Then there is a strategy for $H$ that processes $\ell(I)$ first and uses budget at most $K$.

### 5 Algorithm and Correctness Proof for Theorem 2

In this section we give the algorithm and proof for Theorem 2, that we can solve the bipartite graph ordering problem for classes of graphs with some basic properties. From the previous section it remains to determine how to choose which prime set to process first when there are no positive sets that can be processed.

**Definition 8** Let $I, J$ be prime subsets. We say $I$ potentially is after $J$ for current budget $K$ if

1. $|I| > K$ or,
2. $\text{bg}(\ell(J) \setminus \ell(I)) > K - |\ell(I)| + |N^*(\ell(I))|$

Definition 8 is a first attempt at choosing which prime set to process first. The idea is to consider whether it is possible to process $I$ before $J$. Item 2 in the definition states that $J$ could not be processed immediately after $I$. However, this formula is not sufficient in general because we must consider orderings that do not process $I$ and $J$ consecutively, and we must take into account that for whatever ranking we define on the prime sets the ranking may change as the algorithm processes prime sets. For clarification we have singled out the case when $I$ and $J$ are processed consecutively in the proof of correctness of the algorithm.

If two prime sets $I$ and $J$ are not processed consecutively by the $\text{opt}$ strategy, we should adapt Item 2 of Definition 8 to take into account all vertices that would be processed in between by our algorithm. We call this set of vertices the “Superset” of $J$ with respect to $I$, defined precisely by the recursive Definitions 9 and 10.
Claim 1 Suppose that \( I \) itself a positive set with budget at most \( K \) of processing \( I \). We show that if \( b_g(I) \leq K \), then \( I \) is a minimal positive subset and yet if step 3 were incorrect there would be a way to process \( S \setminus I \) into \( S \setminus J \) from \( B \setminus \text{Superset}_1(J) \) where \( J \) is first according to the lexicographical order of prime sets such that no prime set is before \( J \), according to the ordering in Definition 10. We stop once \( c(I) \) lies in \( \text{Superset}_1(J) \).

Definition 10 For current budget \( K \), we say prime subset \( I \) is after prime subset \( J \) if

1. \(|I| > K| \) or,
2. \( b_g(\text{Superset}_1(J) \setminus c(I)) > K - |c(I)| + |N^*(c(I))| \)

Definition 10 states that \( I \) is after \( J \) if it is too large for the current budget (Item 1) or cannot be processed before \( J \) using the ordering implied by Definitions 9 and 10 (Item 2). Note that if \( I \) is processed right after \( c(I) \) then \( c(I) \) agrees with Definition 8. We point out that Definitions 9 and 10 are recursive, and a naive computation of the ranking would not be efficient. We describe how to efficiently compute the ranking for the classes of graphs of Theorem 2 using dynamic programming in Sections 6 and 7.

The main description of our algorithm is given in Figure 3.

\[
\text{Budget (} \quad H = (B, S), K \quad \text{)}
\]

\[
\text{Input: } H = (B, S) \text{ and budget } K.
\]

\[
\text{Output: true or false.}
\]

1. if \( S = \emptyset \) and \( K \geq 0 \) return true
2. if there is a positive minimal subset \( I \) with \( b_g(I) \leq K \)
   return Budget \( (H(B \setminus I, S \setminus N(I)), K - |I| + |N^*(I)|) \)
3. if the positive set \( I \) with smallest budget has \( (b_g(I) > K) \)
   then return false
4. Let \( I \) be a first lexicographically prime subset with no other prime set before it according to ordering in Definition 10. If no such \( I \) exists then return false
   else return Budget \( (H(B \setminus I, S \setminus N^*(I)), K - |I| + |N^*(I)|) \)
5. if \(|I| > K \) for all prime \( I \subseteq B \) return false

Fig. 3. The algorithm for deciding if \( b_g(H) \leq K \).

The algorithm determines whether \( b_g(H) \leq K \). Note that the exact optimal value can be obtained by using binary search, and since the optimal value is somewhere between 0 and \(|B|\) the exact computation is polynomial if the decision problem is polynomial.

Before considering the running time for the graph classes of Theorem 2 we first demonstrate that the algorithm in Figure 3 decides correctly, though possibly in exponential time, for any \((H = (B, S), K)\).

Lemma 2. For any \( K \) and bipartite graph \( H \), the Budget algorithm (Figure 3) correctly decides if \( b_g(H) \leq K \) or not.

Proof: We show that if \( b_g(H) = K \) then there exists an optimal solution \( \text{opt} \) with budget \( K \) in which subset \( I \) as described in the algorithm is processed first. We use induction on the size of \( B \), meaning we assume that for smaller instances, there is an optimal process that considers the prime subsets according to the rules of our algorithm.

Steps 1 and 5 are clearly correct. The correctness of step 2 follows from Lemma 1. Suppose step 3 were incorrect. Then all positive subsets would have budget above \( K \). Let \( I^+ \) be one such subset and yet if step 3 were incorrect there would be a way to process \( I^+ \) with budget at most \( K \) in \( H \). In that case, we would process some negative set \( I^- \) which somehow reduces the budget of processing \( I^+ \); this can only be so if \( I^- \cap I^+ \neq \emptyset \). In this case Claim 1 states that \( I^+ \cup I^- \) is itself a positive set with budget at most \( K \), a contradiction to the premise of step 3.

Claim 1 Suppose that \( I^+ \) is a positive subset but \( b_g(I^+) > K \) and \( I^- \) is a minimal negative subset where \( b_g(I^-) \leq K \) and \( I^+ \cap I^- \neq \emptyset \). If \( b_g(I^+ \cup I^-) \leq K \) then \( I^+ \cup I^- \) forms a positive subset.
Proof: Let \( X = I^+ \cap J^- \). By the assumption that \( I^+ \) can be processed after processing \( I^- \), we have
\[
|I^+| - |X| \leq K - |I^-| + |N^+(I^-)|.
\]
On the other hand, since \( bg(I^+) > K \) then \( K - |X| + |N^*(X)| < |I^+| - |X| \). From these two we can conclude that:
\[
|N^*(I^-)| > |I^-| - |X|
\]  
(1)

Moreover, because \( I^+ \) is a positive set then \( |N^*(I^+)| \geq |I^+| \). By (1), \( I^+ \) being positive, and the fact that \( |N^*(S \cup T)| \geq |N^*(S)| + |N^*(T)| \) for any \( S \) and \( T \), we have \( |N^*(I^+ \cup I^-)| \geq |N^*(I^-)| + |N^*(I^+) - |I^+| + |I^-| - |X| = |I^+| \).

We are left to verify step 4, so we continue by assuming there are no positive subsets. Suppose the optimal solution \( opt \) processes prime subset \( J \) before \( I \) where \( I \) is the first prime set according to Definition 10 (otherwise we can use induction to say the algorithm is correct). Since there is no positive subset at the beginning, \( opt \) processes \( \ell(J) \setminus J \) after \( J \).

Case 1. Suppose that by induction hypothesis (rules of our algorithm) the \( opt \) would place \( I, \ell(J) \) first in \( H' = (B \setminus \ell(J), S \setminus N^*(\ell(J))) \). In this case \( Superset_1 \setminus \ell(J) \) is just \( \ell(J) \setminus \ell(I) \), and in this case Definitions 8 and 10 coincide.

We show that we can modify \( opt \) to process \( \ell(I) \) first and then \( J \setminus \ell(I) \) next while still using budget at most \( K \). Suppose this is not the case. Now we have the following
\[
(a) \quad K - |\ell(J)| + |N^*(\ell(J))| \geq bg(\ell(I) \setminus \ell(J))
\]
\[
(b) \quad bg(\ell(J) \setminus \ell(I)) > K - |\ell(I)| + |N^*(\ell(I))|
\]

The inequality (a) follows by \( opt \) being an ordering with budget at most \( K \) that processes \( \ell(J) \) first: since \( opt \) processes \( \ell(I) \setminus \ell(J) \) after \( \ell(J) \), the budget must be at least \( bg(\ell(I) \setminus \ell(J)) \). After processing \( \ell(J) \), the inequality (b) follows from the assumption that we cannot process \( \ell(I) \) first and then immediately processing \( J \setminus \ell(I) \). However, this is a contradiction to Definition 10 and the fact that \( I \) is before \( J \) according to Definition 10. We also note that since \( bg(\ell(J)) \leq bg(Superset_1 \setminus \ell(I)) \leq K - |\ell(I)| + |N^*(\ell(I))| \), we can also process the entire \( \ell(J) \) after processing \( \ell(I) \). Therefore we can exchange processing \( \ell(J) \) with \( \ell(J) \) and follow the \( opt \) in the remaining.

Case 2. We are left with the case that \( \ell(J) \) is processed first by \( opt \), and the rules of the algorithm (rule 2 in Definition 10) would process some prime subset \( L \) different from \( I, \ell(J) \) next. This implies that there is some prime subset \( L \) that is considered before the last remaining part of \( I \) in \( B \setminus \ell(J) \). By induction hypothesis we may assume that the \( opt \) processes the prime subsets according to the rule 2 in Definition 10. These would imply \( L \) is in \( Superset_1 \setminus \ell(J) \). At some point \( I \) or the remaining part of \( I \) becomes the first set to process according to the rules of the algorithm and this happens at the last step of the definition of \( Superset_1 \). Therefore, since there is no other prime subset before \( I \) according to Definition 10 we have \( bg(Superset_1 \setminus \ell(I)) \leq K - |\ell(I)| + |N^*(\ell(I))| \). Therefore we can process \( \ell(I) \) first and next \( \ell(J) \setminus \ell(J) \) and then follow \( opt \).

It remains to show that if \( bg(H) \leq K \) then there exists a prime subset \( I \) that is the lexicographically first prime subset with no other prime subset before it according to the ordering in Definition 10. Suppose there exists an ordering for \( H \) with budget at most \( K \) as follows: \( \ell(J_1), \ell(J_2), \ldots, \ell(J_r) \). By induction assume that the Budget Algorithm returns “true” for instance \( H \setminus \{\ell(J) \cup \cup N^*(\ell(J))\} \) with budget \( K - |\ell(J_i)| + |N^*(\ell(J_i))| \) and the output ordering is \( \ell(J_1), \ldots, \ell(J_r) \). Therefore, by Definition 9, \( Superset_1 = \cup_{i=1}^r \ell(J_i) \) for \( 1 \leq i \leq r \). Observe that \( J \) is not after any prime subset by Definition 10 which leads us to have \( J \) as a valid “first” prime subset in \( H \) for the algorithm.

A naive implementation of the algorithm would consider all possible orderings of prime sets to determine the ordering in step 4 of the algorithm, and in the worst-case an exponential number of sets may need to be considered to identify the prime and positive minimal sets. A careful analysis can be taken to show that the running time of the algorithm in the general case is exponential.

In the next two sections we show that for the graph classes of Theorem 2 the running time is polynomial.
6 Polynomial Time Algorithm for Trivially Perfect Bipartite and co-bipartite Graphs

In this section we define trivially perfect bipartite graphs and co-bipartite graphs, and discuss the key properties that are used in our algorithm for solving the bipartite graph ordering problem in these bipartite graphs. In particular, it is possible to enumerate the prime sets of these graphs by looking at a way to construct the graphs with a tree of graph join and union operations.

The subclass of trivially perfect bipartite graphs called laminar family bipartite graphs were considered in [18] to obtain a polynomial time approximation scheme (PTAS) for special instances of a job scheduling problem. Each instance of the problem in [18] is a bipartite graph \( H = (J, M) \) where \( J \) is a set of jobs and \( M \) is a set of machines. For every pair of jobs \( i, j \in J \) the set of machines that can process \( i, j \) are either disjoint or one is a subset of the other. The trivially perfect bipartite graphs also play an important role in studying the list homomorphism problem. The authors of [7] showed that for these bipartite graphs, the list homomorphism problem can be solved in logarithmic space. They were also considered in the fixed parametrized version of the list homomorphism problem in [5].

We call these bipartite graphs “trivially perfect bipartite graphs” because the definition mirrors one of the equivalent definitions for trivially perfect graphs.

**Definition 11 (trivially perfect bipartite graph, co-bipartite graph)** A bipartite graph \( H = (B, S) \) is called trivially perfect or a co-bipartite graph if it can be constructed by applying the following operations.

- A bipartite graph with one vertex is both trivially perfect and a co-bipartite graph.
- If \( H_1 \) and \( H_2 \) are trivially perfect then the disjoint union of \( H_1 \) and \( H_2 \) is trivially perfect. Similarly, the disjoint union of co-bipartite graphs is also a co-bipartite graph.
- If \( H_1 \) and \( H_2 \) are trivially perfect then by joining every sold vertex in \( H_1 \) to every bought vertex in \( H_2 \), the resulting bipartite graph is trivially perfect.
- If \( H_1 \) and \( H_2 \) are co-bipartite graphs, their complete join – where every sold vertex in \( H_1 \) is joined to every bought vertex in \( H_2 \) and every bought vertex in \( H_1 \) is joined to every sold vertex in \( H_2 \) – is a co-bipartite graph.

An example of each type of graph is given in Figure 4.

![Figure 4](https://via.placeholder.com/150)

**Fig. 4.** Each bold line shows a complete connection, i.e. the induced subgraph by \( I_1 \cup P \) is a biclique. In the left figure (trivially perfect) \( I = \{I_1, I_2\} \) and \( J = \{I_2, I_3\} \) are prime sets and \( I \) and \( J \) are sibling. On the right figure (co-bipartite graph) prime sets are \( R_1 = \{J_1, J_2, J_3\} \), \( R_2 = \{J_1, J_2, J_4\} \), \( R_3 = \{J_3, J_4, J_1\} \), \( R_4 = \{J_3, J_4, J_2\} \) are prime sets and for example \( R_1, R_2 \) are sibling.

Theses two classes of bipartite graphs can be classified by forbidden obstructions, as follows.

**Lemma 3 ([7, 10]).** \( H \) is trivially perfect if and only if it does not contain any of the following as an induced subgraph: \( C_6, P_6 \).

\( H \) is a co-bipartite graph iff it does not have any of the following as an induced subgraph \( P_5, Star = (1, 2, 3) \) with vertex set \( V = \{1, 2, 3, 4, 5, 6, 7\} \) and \( E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}, \{3, 7\}\} \), \( Sun(4) \) with vertex set \( V \) and \( E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}, \{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}\} \).

In our algorithm to solve \( bg(H) \) for trivially perfect bipartite graphs and co-bipartite graphs centers around constructing \( H \) as in Definition 11. We view this construction as a tree of operations that are performed to build up the final bipartite graph, and where the leaves of the tree of operations are bicliques. If \( H \) is not connected then the root operation in the tree is a disjoint
union, and each of its connected components is a trivially perfect bipartite graph (resp. co-bipartite graph). If $H$ is connected, then the root operation is a join.

For $H$ a trivially perfect bipartite graph, the join operation must have resulted from one of the following.

- There exist disjoint trivially perfect bipartite graphs $H_1, H_2, \ldots, H_k$ where $H \setminus H_1$ has connected components $H_2, \ldots, H_k$ and the sold vertices in $H_1$ are adjacent to all bought vertices in every $H_i$, $2 \leq i \leq k$ and no bought vertex in $H_1$ is adjacent to any sold vertex in $H_2 \cup H_3 \cup \cdots \cup H_k$. Furthermore we assume that $H_1$ is maximal. In this case we say $H_1$ is the parent of $H_2, \ldots, H_k$ and each $H_i, H_j$, $2 \leq i < j \leq k$ are siblings$^6$.

- There exist disjoint trivially perfect bipartite graphs $H_1, H_2, \ldots, H_k$ where $H \setminus H_k$ has connected components $H_1, \ldots, H_{k-1}$ and the sold vertices in $H_i$, $1 \leq i \leq k - 1$ are all adjacent to bought vertices in $H_k$. Furthermore we assume that $H_k$ is maximal. If $H_k$ does have sold vertices then $H_k$ is the parent of $H_1, \ldots, H_{k-1}$ and each $H_i, H_j$, $1 \leq i < j \leq k - 1$ are siblings. Otherwise ($H_k$ does not have sold vertices) $H_1 \cup H_k, H_2 \cup H_k, \ldots, H_{k-1} \cup H_k$ are siblings.

For $H$ a co-bipartite graph, the join operation must have resulted from joining co-bipartite graphs $H_{11}, H_{12}, \ldots, H_{1k}, H_{21}, H_{22}, \ldots, H_{2r}$ where

- $H \setminus H_1$ ($H_1 = H_{11} \cup H_{12}, \ldots, \cup H_{1k}$) has connected components $H_{21}, \ldots, H_{2r}$
- $H \setminus H_2$ ($H_2 = H_{21} \cup H_{22}, \ldots, \cup H_{2r}$) has connected components $H_{11}, H_{12}, \ldots, H_{1k}$
- Sold vertices in $H_1$ ($H_2$) are adjacent to all bought vertices in $H_2$ ($H_1$).
- Each $H_{11} \cup H_{21}, H_{12} \cup H_{22}, \ldots, 1 \leq i < j \leq r$ are siblings and each $H_{21} \cup H_{11}, H_{11} \cup H_{1j}, 1 \leq i < j \leq k$ are siblings and each $H_{22} \cup H_{11}, H_{12} \cup H_{2j}, 1 \leq i < k, 1 \leq j < r$ are siblings.

By further decomposing each $H_i$ we obtain a tree $T$. The leaves of $T$ are bicliques in $H$. Note that for trivially perfect bipartite graphs a prime subset necessarily is the bought part of a biclique that is a leaf in the tree $T$; however, not all leaves have a prime subset as its bought part. For co-bipartite graphs, the prime sets are precisely the set of siblings of the root join operation.

**Lemma 4.** Given a trivially perfect bipartite graph (resp. co-bipartite graph) $H$ with $n$ vertices, a decomposition tree for $H$ can be found in $O(n^3)$.

**Proof:** First suppose $H = (A, B, E)$ is a trivially perfect bipartite graph with $n$ vertices. If $H$ is not connected then the root of tree $T$ is $H$ and each child of root is a connected component of $H$ and root has label union. If $H$ is connected then we proceed as follows. Let $1 \leq m \leq n$ be a maximum index such that the following test passes. Let $B_2$ be the set of vertices in $B$ which have degree at least $m$ and let $B_1 = B \setminus B_2$. Let $A_1$ be the set of all vertices in $A$ that are common-neighbors of all the vertices in $B_2$. If $|A_1| < m$ then the test fails. Now there exists a vertex $v \in B_1$ such that $N(B_1) \not\subseteq A_1$ then the tests fails. If the test passes then let $A_2 = A \setminus A_1$ and let the root of $T$ be $H$ with label * and the left child of $H$ is $H[B_1 \cup N(B_1)]$ and the right child of the root would be $H_2 = H \setminus H_1$. Note that by the definition of trivially perfect bipartite graphs. If the test fails for every $m$ then clearly $H$ is not trivially perfect.

We continue the same procedure from each node of the tree until each vertex each node becomes a biclique. Node that $T$ has at most $n$ nodes. For a particular $m$, checking all the condition of the test takes $O(n^2)$. Therefore the whole procedure takes $O(n^3)$. For co-bipartite graph $H$ the polynomial time decomposition is given in [10].

In what follows we show that there is a subclass of trivially perfect bipartite graphs that are also circle bipartite graphs. A bipartite graph $H = (B, S)$ is called a chain graph if the neighborhoods of vertices in $B$ form a chain, i.e, if there is an ordering of vertices in $B$, say $w_1, w_2, \cdots, w_p$, such that $N(w_1) \supseteq N(w_2) \supseteq \cdots \supseteq N(w_p)$.

It is easy to see that the neighborhoods of vertices in $S$ also form a chain. *Chain graphs* are subsets of both trivially perfect bipartite graphs and circle bipartite graphs. Any chain graph can be visualized as what is depicted in Figure 5(a), and the corresponding RNA model for the bipartite graph ordering problem looks like Figure 5(b).

$^6$ Siblings are subgraphs such that some stage of the Budget Algorithm must decide which of the siblings to process next.
Theorem 12. For trivially perfect bipartite graphs and co-bipartite graphs \( H \) with \( n \) vertices the Budget algorithm (Figure 3) runs in \( O(n^4) \).

Proof: When \( H \) is trivially perfect or a co-bipartite graph, Lemma 4 states that we can compute the decomposition tree for \( H \) in time \( O(n^3) \). The first step of the Budget algorithm deals with the positive minimal sets. Let \( I \) be a positive minimal set and let \( H' \) be the induced subgraph of \( H \) by \( I \cup N^*(I) \). If \( H' \) is not connected then there is at least one connected component of \( H' \) that is positive, a contradiction to minimality of \( I \). We continue by assuming \( H \) is connected.

According to the decomposition of \( H \) there are \( H_1, H_2 \) such that \( H_1 \) and \( H_2 \) are trivially perfect (resp. co-bipartite graphs) and \( H_1 \) was joined to \( H_2 \). For \( H \) a trivially perfect bipartite graph, the side \( H_i \) with its bought vertices \( B \) attached to the sold vertices \( S \) of the other must be processed first; thus any positive set must include either a positive part of \( H_i \) or all of \( H_i \) together with a positive part of the other side. For \( H \) a co-bipartite graph, any positive set must include either \( H_1 \) or \( H_2 \) together with a part of the other. In either case, we can traverse the decomposition tree to search for a positive set.

We now consider the determination of the ordering in step 4 when there is no positive set.

In order to keep track and answer queries such as \( bg(c\ell(J)) \leq w \) we create a dynamic programming table and fill out the entry of that table for each such \( J \). Since all prime sets \( J \) must be leaves of the decomposition tree, there are only a polynomial number of prime sets to consider and by Lemma 4 we can determine what they are. It takes \( O(n) \) time to traverse the decomposition tree.

For a given prime set \( J \), computing \( c\ell(J) \) takes at most \( O(n) \) time. Thus the table has polynomial size.

The main task of the algorithm is determining in step 4 the order of prime sets \( I \) and \( J \) according to Definition 10. The following claims simplify the possible interactions that might exist between primes \( I \) and \( J \) in terms of the order.

Claim 2 Let \( H \) be a trivially perfect bipartite graph and let \( I \) and \( J \) be two prime subsets in subgraph \( H' \) of \( H \) where \( H' \) does not have any positive proper subset in \( H' \). If \( I \subset c\ell(J) \) then \( I \) and \( J \) are siblings. If \( I, J \) are not siblings then \( c\ell(I) \cap c\ell(J) = \emptyset \).

Proof: Let \( I_1, I_2 \) be the first two subsets in \( c\ell(I) \) according to the definition. Consider \( N_1 = N_{H_1}(I_1) \) where \( H_1 = H' \setminus (I \cup N^*(I)) \). Since there is no positive subset in \( H' \) one of the following happens:

(a) every sold vertex in \( N_1 \) is adjacent to every in \( I \)
(b) there exists a subset \( N'_1 \) of \( N_1 \) where every sold vertex in \( N'_1 \) is adjacent to every vertex in \( I \).

Note that by definition no vertex \( x_1 \in N^*(I) \) is not adjacent to any vertex in \( I_1 \cup I_2 \) and no vertex \( x_2 \in N_1 \) is adjacent to any vertex \( i_2 \) of \( I_2 \). Let \( x \) be a vertex in \( N^*(I_2) \) in \( H_1 \setminus (I_1 \cup N_1) \) that is adjacent to \( i_1 \in I_1 \). Note that \( x \) exists according to (a,b). If \( x \) is not adjacent to a vertex \( i \in I \) then \( i_1, x, i_1, x_2, i, x_1 \) induces \( P_8 \), a contradiction. Therefore \( x \) is adjacent to every vertex in \( I \). This means that \( I_2 \) and \( I \) are siblings. By continuing this argument we conclude that \( I \) and
Claim 3 Let $H$ be a co-bipartite graph and let $I$ and $J$ be two prime subsets in subgraph $H'$ of $H$ where $H'$ does not have any positive proper subset. If $I \subset \ell(J)$ then $I, J$ are siblings. If $I, J$ are not siblings then $\ell(I) \cap \ell(J) = \emptyset$.

Proof: It follows from the definition of co-bipartite graphs.

Note that for a given two prime sets $I, J$, computing $bg(Superset_1(J))$ takes $O(n^2)$ time as it involves computing $\ell(J)$ and then adding other created prime sets in the decomposition tree according to Definition 10. When we compare two prime sets $I, J$, if $I, J$ are siblings then we compare them according to the rules in Definition 8, and this process takes $O(n)$ time. Otherwise $I$ and $J$ are not siblings (and by the claims $\ell(I) \cap \ell(J) = \emptyset$, and hence from disjoint components of $H$), and we use the following observation.

Observation 13 Let $H_1, H_2$ be two connected components of $H$ (i.e. $H_1 \cap H_2 = \emptyset$) and suppose the Budget algorithm provides an optimal strategy for computing the budget for $H_1$ and $H_2$ separately. Let $I_1, J_1$ be the first prime sets in $H_1, H_2$, respectively, according to Definition 10. $J_1$ is after $J_2$ in $H$ if $bg(H_2) > K - |\ell(J_1)| + |N^*(\ell(J_1))|$. Otherwise $J_1$ comes before $J_2$ if $J_1$ is potentially before $J_2$.

Note that if $bg(H_2) > K - |\ell(J_1)| + |N^*(\ell(J_1))|$ then clearly the optimal process $opt$ does not start with $J_1$. Thus $opt$ must start with $J_2$ when it wants to process $H_1 \cup H_2$.

Now let $H_1$ and $H_2$ be two connected components of $H$ in $H \setminus (\ell(J) \cup N^*(\ell(J)))$ where $H_1$ contains $I_1 = I \setminus \ell(J)$ and $I_1$ comes first according to Definition 10 (for simplicity we may assume $I_1$ comes first). This means we should process $I_1$ first in the $opt$ process for $H_1$. Now we construct the $Superset_1(J)$ according to the following:

Let $I_2$ be a first prime subset according to the ordering in Definition 10 in $H_2$. If $I_2$ is before $I_1$ according to the ordering in Observation 13 then place $I_2$ into $Superset_1(J)$ and set $H_2 \leftarrow H_2 \setminus (\ell(I_2) \cup N^*(\ell(I_2)))$. Else stop.

Note that if $I, J$ are not siblings we can answer the query $bg(Superset_1(J) \setminus \ell(I)) \leq w$ using the above rules which can take $O(n^2)$ time. Overall we may end up spending $O(n^3)$ for all possible pairs of prime sets $I, J$ (we use Observation 13 for comparisons and only need to compare pairs from disjoint components). Each basic step of the Budget algorithm involves finding one prime set to start with. Since there are $O(n)$ nodes in the decomposition tree, the overall running time is $O(n^4)$.

7 Polynomial Time Algorithm for Convex Bipartite Graphs

A bipartite graph $H = (B, S)$ is called convex if there exists an ordering $\prec$ of the vertices in $B$ such that the neighborhood of each vertex in $S$ consists of consecutive vertices in $\prec$. We refer to a set of consecutive vertices in such an ordering as an interval. Figure 2 is an example of a convex bipartite graph. Note that the class of circle bipartite graphs $G = (X, Y)$, for which obtaining the optimal budget is NP-complete, contains the class of convex bipartite graphs.

Let $H = (B, S)$ be a convex bipartite graph and $\prec$ be an ordering on $B$ as described above. We show the following lemma, which we use to prove Theorem 15.

Lemma 5. Let $v$ be the rightmost vertex in $B$ according to $\prec$. Let $H_1 = (B - v, S - N(v))$ be a convex bipartite graph without positive minimal sets. Given the optimal ordering according to the Budget algorithm (Figure 3) for $H_1 = (B - v, S - N(v))$, $Superset_1(J)$, for all $I, J \in H = (B, S)$, can be computed in time $O(n^3)$.

Proof: Here we consider two cases. The first one is when $N^*(v) = \emptyset$ and second is when $N^*(v) \neq \emptyset$.

Case 1. Suppose $N^*(v) \neq \emptyset$. By the first step of the Budget Algorithm (Figure 3), we remove $v$ and $N^*(v)$. Note that after it we do not decrease our capital.
Let $I$ be the rightmost prime interval in $H' = (B - v, S - N^*(v))$ and $J \neq I$ be an interval in $H' = (B - v, S - N^*(v))$. Without lose of generality, assume $H'$ does not have a positive minimal set. Note that $\text{Superset}_1(J)$ in $H_1$ and $H'$ might be different from each other.

**Case 1.a** Consider computing $\text{Superset}_1(J)$. The first set to be added to $\text{Superset}_1(J)$ is $c\ell(J)$. After that we compare the first prime set $J_1 \in H''$ where $H'' = H' \setminus \text{Superset}_1(J)$, assuming that we do have the order of prime sets to process in $H''$ (this is obtained from the previously computed optimal ordering in $H''$). Therefore, $I$ should be compared with $J_1$ in $H''$ according to the rules of Definition 10. Since we need to compare at most $n$ sets ($n$ sets are added into $\text{Superset}_1(J)$) and each comparison takes $O(n)$ therefore the overall required time would be $O(n^2)$.

**Case 1.b** Consider computing $\text{Superset}_1(I)$. As a result of the recursive nature of the definition for $\text{Superset}$, it requires spending more time in order to compute $\text{Superset}_1(I)$. Note that the first prime set added to $\text{Superset}_1(I)$ is $c\ell(I)$. Afterwards, we are left with an instance of the problem in which the neighbourhood of at most $O(n)$ intervals have been changed and because $I$ was the rightmost interval and convexity of the graph, it requires processing the rightmost interval in the remaining instance to those changes occurred. To wrap up what we discussed we have the following recursive formula to find $\text{Superset}_1(I)$. Let $T(n)$ be the time complexity function assigned to finding $\text{Superset}_1(I)$ in $H_1$. Observe that since the graph is convex bipartite then neighbourhood of at most $n$ intervals has been changed after removing $v \cup N^*(v)$ (if $N^*(v) \neq N(v)$). Therefore we have the following,

$$T(n) \leq T(n - 1) + O(n^2)$$

where $T(n - 1)$ denotes the time complexity function for the new instance after removing $c\ell(I)$ and $O(n^2)$ stands for the time we need to spend for computing $\text{Superset}_1(J)$ (Case 1.a). The above recursion leads us to $O(n^3)$ time complexity.

**Case 1.c** Consider computing $\text{Superset}_1(J')$ where $J \neq J' \neq I$. A similar argument to the two above cases can be used to show that $\text{Superset}_1(J')$ can be computed in time $O(n^3)$.

**Case 2.** If $N^*(v) = \emptyset$. This case is similar to the above case. The difference is that, here it is not clear whether we should process $v$ at first or not. In the similar scenario, in $H$ we might have different primes with different neighbourhoods than in $H_1$. Note that the number of these primes is $O(n)$ and the convexity implies an ordering process from right to left on these new primes to have different neighbourhood than ones in $H$. Similar argument yields us $O(n^3)$ time complexity.

**Observation 14** Suppose $\text{Superset}_1(J)$, for arbitrary prime intervals $J, J' \in B$, consists of $c\ell(J_0 = J) \cup c\ell(J_1) \cup \cdots \cup c\ell(J_r = J')$ while $c\ell(J_i)$ is added to $\text{Superset}_1(J)$ before $c\ell(J_i)$ for $0 \leq i < j \leq r$. Then, $\text{Superset}_1(J) = c\ell(J_0 = J) \cup c\ell(J_1) \cup \cdots \cup c\ell(J_r)$.

Since there are potentially $O(n^2)$ prime intervals then we have $O(n^4)$ pairs of them. In fact the above observation shows that the total running time needed to compute $\text{Supersets}$ of all pairs of prime intervals is $O(n^3)$.

**Theorem 15.** There exists an algorithm which solves the bipartite graph ordering problem on convex bipartite graphs in time $O(n^4)$.

**Proof:** Let $H = (S, B)$ be a convex bipartite graph with an ordering on its vertices as described above. We start from the rightmost vertex $v$ in $B$ and move to the left, iteratively determining $\text{Superset}_1(J)$ and bg in larger subgraphs of $H$ from the right. We assume the optimal ordering for graph $H_1 = H \setminus \{v \cup N(v)\}$ has been computed according to the Budget algorithm (Figure 3). We show how to find an optimal ordering for $H$ following the rules of the Budget algorithm.

The second step of the Budget algorithm consists of finding all positive minimal sets. For convex bipartite graphs, prime sets, the closure of a prime set ($c\ell(I)$), and any positive minimal set is an interval; thus we must keep track of the budget needed to process intervals in $B$, and there are only polynomially many of these to consider (in particular, $O(n^2)$ where $B$ has $n$ vertices). Considering prime sets and closures of prime sets, when we consider an interval $J$ we need to consider $N^*(J)$ together with vertices of $S$ that are not initially in $N^*(J)$ but are initially in $N(J)$ where $N(J) \cap N(J') \neq \emptyset$. This is because after processing $J' \setminus J$ we may need to determine
the budget for \( J \) in the new instance and here the neighborhood of \( N^*(J) \) contains new vertices. However, the number of these new intervals is at most \( O(n^2) \). This task is in fact computing the budget for a proper sub-interval \( J \) in \( B \) together with the appropriate set of vertices in \( S \) which may need to be considered in \( N^*(J) \). From now on we assume we can answer the queries such as \( bg(cℓ(J)) \leq w \) for a given \( w \) using a dynamic programming table.

Consider processing a positive minimal set. According to Definition 6, it does not have any proper positive subset. Therefore, it is the same as the case when we have a convex bipartite graph without any positive prime interval and no positive closure set (Definition 7). Note that once we have the Supersets for different prime intervals it is easy to verify whether a positive minimal set can be processed with the current budget.

We continue analyzing step 4 of the algorithm assuming there is no positive prime interval and no positive closure set in \( H_1 \). The main work here is finding Supersets for \( I \) and \( J \) being intervals in \( B \). Once the Supersets, for all \( I, J \in B \), are computed we can compare them according to Definition 10 and apply step 4 of the Budget algorithm. It is worth mentioning that Superset in \( H_1 \) might be different from Superset in \( H \). Lemma 5 shows that the latter can be done derived from the former in time complexity \( O(n^3) \) for all pairs of prime intervals.

Since we move from right to left on \( H \), which has \( |B| = n \), we repeat the above \( n \) times, and finding the optimal ordering for \( H \) takes \( O(n^3) \) time. Note that processing intervals of vertices might lead us to disconnected instances; however, the above argument works for disconnected convex graphs.

8 Extension to a more general class of bipartite graphs

Let \( H = (B, S) \) be a bipartite graph. Suppose there exists a tree \( T \) with a bounded number of leaves, say \( d \) leaves. \( B = V(T) \) and each vertex in \( S \) is adjacent to a set of vertices in \( T \) that forms a sub-tree of \( T \). Note that in the case convex bipartite graphs tree \( T \) is just a path. Each prime subset in \( B \) is a sub-tree of \( T \) and since \( T \) has at most \( d \)-leaves, there are at most \( n^d \) subsets to consider and when \( d \) is fixed there are polynomially many of these prime subsets.

The spine of \( T \) is a longest path in \( T \). Let \( S = v_1, v_2, \ldots, v_q \) be a spine of \( T \). The index of a vertex \( x \) in \( T \) is the smallest \( s \) such that \( v_s \) is the closest vertex to \( x \). For two vertices \( x \) and \( y \) of \( T \) we say \( x \) is before \( y \), (we write \( x \prec y \)) if the index of \( x \) is less than the index of \( y \). When \( x \) and \( y \) have the same index, then \( x \prec y \) if \( x \) is on the spine \( S \), or if the path connecting \( x \) and \( y \) does not cross the spine and \( x \) is closer to \( v_s \) than \( y \) (note that the \((x,y)\)-path is unique since \( T \) is a tree). Now we take linear extension of this partial ordering and get a total ordering \( <_T \).

The index of sub-tree \( P \) is the index of the vertex with the smallest index among the vertices in \( P \). We say sub-tree \( P \) is before sub-tree \( Q \) and we write \( P < Q \) if the smallest vertex in \( P \) with respect to \( <_B \) precedes that of \( Q \), or if the two sub-trees have the same smallest vertex and the greatest vertex of \( P \) with respect to \( <_B \) lies inside \( Q \). Now we take a linear extension of the ordering of sub-trees and we get a total ordering \( <_T \) for sub-trees. Now this ordering is an ordering for the subsets including the prime subsets. Now for a prime subset \( I \) we can compute the \( cℓ(I) \) and also we can compute Superset for a prime subset \( J \) in polynomial time. These indicate that we can deploy the same Budget algorithm and find the budget for \( H \) in polynomial time.

9 Future Work and Open Problems

We have defined a new scheduling or ordering problem that is natural and can be used to model processes with precedence constraints. As with any optimization problem there are many avenues of attack. In this work we have focused on determining for which classes of graphs the bipartite graph ordering problem can be solved in polynomial time. Our ultimate goal in this direction is a dichotomy classification of polynomial cases and NP-complete cases. The algorithm in the proof of Theorem 2 finds the optimal budget for all graphs \( H \), and the algorithm was shown to run in polynomial time for the classes of graphs mentioned in Theorem 2. We pose the question whether the algorithm can be the basis of a dichotomy theorem: are there classes of graphs which can be solved in polynomial time but for which our algorithm does not run in polynomial time?
As with all optimization problems the bipartite graph ordering problem can also be studied from a number of other angles, including approximation and hardness of approximation, fixed parameter algorithms, and faster exponential-time algorithms. A particular graph class to consider in each of these areas is that of circle bipartite graphs, because these graphs are of particular interest in the application to molecular folding [9, 16, 19].

Acknowledgments
We would like to thank Pavol Hell, Ladislav Stacho, Jozef Hales, Cedric Chauve and Geoffrey Exoo for many useful discussions.

References

[1] F. Alizadeh, R.M. Karp, L.A. Newberg, and D.K. Weisser. Physical mapping of chromosomes: a combinatorial problem in molecular biology. Algorithmica 13(1/2): 52–76 (1995).

[2] C. Ambühl, M. Mastrolilli, N. Mutsanas, O. Svensson: On the approximability of single-machine scheduling with precedence constraints. Math. Oper. Res. 36(4): 653–669 (2011).

[3] C. Ambühl, M. Mastrolilli, and O. Svensson. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In Proceedings of FOCS : 329–337 (2007).

[4] H. L. Bodlaender, F. V. Fomin, A. M.C.A. Koster, D. Kratsch and D.M. Thilikos. A note on exact algorithms for vertex ordering problems on graphs. Theory Comput. Syst 50(3):420–432 (2012).

[5] R. H. Chitnis, L. Egri, and D. Marx. List H-coloring a graph by removing few vertices. In Proceedings of ESA 313–324 (2013).

[6] J. R. Correa, A. S. Schulz. Single-machine scheduling with precedence constraints. Math. Oper. Res. 30(4): 1005–1021 (2005).

[7] L. Egri, A. d Krokhin, B. Larose and D. Tess. The Complexity of the list homomorphism problem for graphs. Theory of Computing Systems, 51(2):143–178 (2012).

[8] C. Flamm, I. L. Hofacker, S. Maurer-Stroh, P. F. Stadler and M. Zehl., Design of multistable RNA molecules. RNA 7 (02): 254–265 (2001).

[9] M. Geis, C. Flamm, M. T. Wollinger, A. Tanzer, I. L. Hofacker, M. Middendorf, C. Mandl, P. F. Stadler and C. Thurner. Folding kinetics of large RNAs. J. Mol. Biol. 379(1), 160–173 (2008).

[10] V. Giakoumakis, J.M. Vanherpe. Bi-complement reducible graphs. Adv. Appl. Math. 18:389–402 (1997).

[11] M. Habib, R. M. McConnell, C. Paul, and L.Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theor. Comput. Sci. 234(1-2): 59–84 (2000).

[12] K. Khodamoradi, R. Krishnamurti, A. Rafiey, G. Stamoulis. PTAS for Ordered Instances of Resource Allocation Problems. Proceeding of FSTTCS 461–473 (2013).

[13] W. Lu, and W. Hsu. A test for the consecutive ones property on noisy data - application to physical mapping and sequence assembly. J. Comput. Biology 10(5): 709–735 (2003).

[14] J. Manuch, C. Thachuk, L. Stacho and A. Condon. NP-completeness of the direct energy barrier problem without pseudoknots. In Proceedings of the 15th Intl. Meeting on DNA Computing and Molecular Programming (DNA15), 106–115 (2009).

[15] R. M. McConnell. A certifying algorithm for the consecutive-ones property. SODA 768–777 (2004).

[16] S. R. Morgan and P. G. Higgs. Barrier heights between ground states in a model of RNA secondary structure. J. Phys. A: Math. Gen. 31(14), 3153 (1998).

[17] R. H. Möhring, M. Skutella, F. Stork: Scheduling with AND/OR precedence constraints. SIAM J. Comput. 33(2): 393–415 (2004).

[18] G. Muratore, U. M. Schwarz, and G. J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. Oper. Res. Lett., 38(1):47–50 (2010).

[19] C. Thachuk, J. Manuch, A. Rafiey, L. Mathieson, L. Stacho and A. Condon. An algorithm for the energy barrier problem without pseudoknots and temporary arcs. In Pacific Symposium on Biocomputing, 15:108–119 (2009).

[20] G. J. Woeginger. On the approximability of average completion time scheduling under precedence constraints. Discrete Applied Mathematics, 131(1):237–252 (2003).