mathematics

MDPI

Article

Clique Search in Graphs of Special Class and Job Shop Scheduling

Sándor Szabó [1] and Bogdán Zaválnij [2,*]

[1] Institute of Mathematics and Informatics, University of Pécs, 7622 Pécs, Hungary; sszabo7@hotmail.com
[2] Rényi Institute of Mathematics, 1053 Budapest, Hungary
* Correspondence: bogdan@renyi.hu

Abstract: In this paper, we single out the following particular case of the clique search problem. The vertices of the given graph are legally colored with k colors and we are looking for a clique with k nodes in the graph. In other words, we want to decide if a given k-partite graph contains a clique with k nodes. The maximum clique problem asks for finding a maximum clique in a given finite simple graph. The problem of deciding if the given graph contains a clique with k vertices is called the k-clique problem. The first problem is NP-hard and the second one is NP-complete. The special clique search problem, we propose, is still an NP-complete problem. We will show that the k-clique problem in the special case of k-partite graphs is more tractable than in the general case. In order to illustrate the possible practical utility of this restricted type clique search problem we will show that the job shop scheduling problem can be reduced to such a clique search problem in a suitable constructed graph. We carry out numerical experiments to assess the efficiency of the approach. It is a common practice that before one embarks on a large scale clique search typically one attempts to simplify and tidy up the given graph. This procedure is commonly referred as preconditioning or kernelization of the given graph. Of course, the preconditioning or kernelization is meant with respect to the given type of clique search problem. The other main topic of the paper is to describe a number of kernelization methods tailored particularly to the proposed special k-clique problem. Some of these techniques works in connection with the generic k-clique problem. In these situations, we will see that they are more efficient in the case of k-partite graphs. Some other preconditioning methods applicable only to k-partite graphs. We illustrate how expedient these preconditioning methods can be by solving non-trivial scheduling problems to optimality employing only kernelization techniques dispensing with exhaustive clique search algorithms altogether.

Keywords: job shop scheduling; kernelization; k-clique; struction

MSC: 68M20; 90B35; 90B36; 05C85

Citation: Szabó, S.; Zaválnij, B. Clique Search in Graphs of Special Class and Job Shop Scheduling. *Mathematics* **2022**, *10*, 697. https://doi.org/10.3390/math10050697

Academic Editors: Miklós Krész and Frank Werner

Received: 18 December 2021
Accepted: 15 February 2022
Published: 23 February 2022

1. Introduction

This paper is part of a larger on going project of the authors. Namely, we are considering different combinatorial optimization problems that can be reduced to a clique search applied to a specially constructed auxiliary graph. In this sense, we use the clique search framework as a mathematical programming modeling tool and we use a clique search solver as a general solver. The situation is is analogous to using linear programs with integer variables for modeling and using LP solvers as generic solvers. The case of the satisfiability problem or the constraint programming is very similar. Previous publications showed that we can take advantage of the clique search approach in cases such as solving the Keller's conjecture [1], different type of graph coloring [2], hypergraph and mixed hypergraph coloring [3] and molecular docking problems [4]. For more details see [5].

In this paper, only finite simple graphs will appear, that is, the graphs have finitely many nodes and finitely many edges. They do not have any loop or double edge. Let $G = (V, E)$ be a finite simple graph. Here V stands for the set of vertices of G and E is the set of edges of G. Because G does not have any loop or double edge we may identify

E with a subset of the Cartesian product $V \times V$. A subset Δ of V is called a clique of G if each two distinct nodes of Δ are adjacent in G. The cardinality of the set Δ is referred to as the size of the clique. If $|\Delta| = k$ we will say that Δ is a k-clique in G. For each finite simple graph G there is an integer k such that the graph G has a k-clique but G does not have any $(k+1)$-clique. This well defined integer k is called the clique number of G and is denoted by $\omega(G)$. We will use the notation $N(a)$ for the set of neighbors of the node a, that is, $N(a) = \{x : x \in V, \{a, x\} \in E\}$.

It is customary to assign colors to the nodes of a graph G such that the assignment satisfies the following conditions. Each node has exactly one color and adjacent nodes never receive the same color. We refer to such coloring as a legal coloring of the nodes of the graph G. For each finite simple graph G there is an integer k such that the nodes of G can be legally colored using k colors but the nodes of G cannot be legally colored using $(k-1)$ colors. This well defined integer k is called the chromatic number of G and is denoted by $\chi(G)$. A finite simple graph is called a k-partite graph if its nodes can be legally colored using k colors. Note that for the clique and the chromatic numbers the inequality $\omega(G) \leq \chi(G)$ holds.

A finite simple graph G is called a perfect graph if the equation $\omega(H) = \chi(H)$ holds for each induced subgraph H of G. Determining $\omega(G)$ is a computationally demanding problem. Namely, it is an NP-hard problem. Similarly, for a general finite simple graph G computing $\chi(G)$ is an NP-hard problem. It is a remarkable property of the class of perfect graphs that the problem of computing their clique and chromatic numbers belong to the P complexity class.

1.1. Problem Definition

In this paper, in connection with scheduling problems we will be interested in the following clique search problem. The nodes of a finite simple graph G are legally colored with k colors and we want to decide if G admits a k-clique or not. If G has a k-clique, then we may conclude that $\omega(G) = \chi(G)$ holds. This itself does not imply that G is a perfect graph. The problem of deciding if there is a k-clique present in G is still an NP-complete one as the problem that $\chi(G) \leq k$ can be reduced to an s-clique problem in connection with a suitable constructed s-partite graph [2]. However, the k-clique problem in k-partite graph is less unwieldy compared with the case of general k-clique problem. We will present kernelization results that take advantage of the special setting and will make the clique search computationally more efficient for the special class of k-partite graphs.

In this paper, we restrict our attention only to a handful of the so-called clique search problems. We describe them in a formal manner here. The maximum clique problem, which is an optimization problem, asks for determining the clique number of a given graph. It may happen that we would like to locate a clique with maximum size in the graph or we wish to list all maximum cliques.

The k-clique problem is decision problem. Namely, given a finite simple graph G and a positive integer k. Decide if G contains a clique of size k. Again it may happen that one would like to exhibit a k-clique of G or list all k-cliques of G.

The various clique search problems are not independent of each other. For instance, the maximum clique problem can be polynomially reduced to the k-clique problem and conversely. The clique search problems are motivated by real life applications and they have theoretical significance as well. For further details see [6].

1.2. Structure of the Paper

The paper can be divided into three parts. In the Section 2, we present commonly used preconditioning methods. Many of these techniques are part of the mathematical folklore and it is not possible to find where they were first mentioned. For the convenience of the reader and for easier reference we use this section to state some of the results in a formal manner. We describe different graph kernelization methods based on legal coloring of the nodes, on dominance relations and on transformations of edges into nodes. Typically, we

carry out inspections by scanning nodes or edges and depending on the result we delete nodes or edges from the graph. We pay particular attention to the special case when the graph is k-partite. The Section 3 is about the so-called struction graph transformation. At this juncture we do can name the source of the basic idea. The struction construction is due to [7,8]. We particularized the struction construction to the special case of k-partite graph. The results of this section are important and new.

In the Section 4, we describe a graph construction. This construction makes possible to reduce the job shop scheduling problem to a clique search problem. We will show that the constructed auxiliary graph is k-partite and the relevant clique search problem is a k-clique problem. In the Section 5, we carry out numerical experiments. There are no theoretical tools to estimate the running times and this is why we have to follow the common practice of the trade of resorting on numerical experiments in connection with well chosen instances from the literature. We categorize these instances as small, medium and large instances. We show that there are large instances that are solvable to optimality by using the proposed preconditioning methods alone. In the last section, we draw conclusions.

2. Kernelization

Before embarking on a large scale clique search it is advisable to carry out a thorough inspection of the original graph to detect deletable nodes and edges. We may express these type of results by saying that the original graph is transformed to a new simpler graph. This suggests the line of inquiry to try to use other type of transformations of the underlying graph. Our aim is to reduce the number of the nodes of the transformed graph while keeping the size of the sought after clique or reducing the size of this clique by one.

Kernelization is an extensively studied field (see, for example, [9–12]). We would like to point out certain aspects that deserve closer attention. They concern the edge density of the underlying graph. Firstly, the commonly used preconditioning methods are essentially useful for extremely dense graphs in case of the maximum clique problem. Similarly, the traditional kernelization techniques are mainly useful for extremely sparse graphs in case of the maximum independent set or minimum vertex cover problems. T. Akiba and Y. Iwata [9] draw the conclusion that these kernelization methods are mostly futile for moderately dense graphs. In the test graph instances, we use the edge density of the graphs are high but not to such an extent that the mentioned proposed algorithms would benefit from it. Secondly, most of the proposed kernelization methods are a simplified versions of the struction transformation [7,8], tailored for specific subgraph configurations. These specific configurations are rare in the test cases we worked with. Moreover, these subcases can be considered as special cases of the struction transformation.

We decided to focus here to less sophisticated kernelization methods. These methods are not the most powerful but they can be used widely. The outline of the kernelization work flow is the following. Firstly, we will look at a legal coloring of the nodes of the graph and explore if this coloring can contribute to preconditioning. Secondly, we will employ a so-called dominance relation together with its various extensions. Thirdly, we introduce a simple edge to node transformation.

Certain facts about kernelization are stated formally as "properties". We have to point out here (as we already have done earlier) that the presented "properties" are mostly part of the mathematical folklore and from this reason it is not possible to attribute them to any specific source. These are Properties 1, 2 and 5. We definitely do not claim that these results are our original contributions. We state them simply for manageable reference and for the convenience of the reader. Other "properties", such as Properties 3, 4, 6 and 7, are direct extensions of folklore results and again they cannot be referenced. In addition it would be overly pedantic to point out sources of results what the reader can verify after a short contemplation. Three kernelization techniques are new contributions of the paper. These are the β transformation, the ideas described as red-black edges, and the results about struction transformation particularized to k-partite graphs.

2.1. Color Indices

If a finite simple graph G has an isolated node, that is, a node that is not adjacent to any other nodes of G, then clearly this node can be ignored when we are looking for a k-clique in G. Provided of course that we are not looking for a 1-clique in G. Similarly, if the degree of a node is too small compared to k it cannot be a node of a k-clique in G. We are extending this preconditioning observation to the case when the nodes of G are legally colored, using the fact that a k-clique must be colored by exactly k colors.

Suppose that the nodes of a finite simple graph G are legally colored using k colors and $C_1, \ldots, C_k$ are all the color classes of this coloring.

Definition 1. *The color index of a node v of G (with respect to a legal coloring of the nodes of G) is the number of color classes C_i that contain at least one node adjacent to v.*

Note that if the color index of a node v is less than $k-1$, then v cannot be a node of a k-clique in G. We state this observation in a more formal manner.

Property 1. *If the colors index of v is at most $k-2$, then v can be deleted from G without losing any k-clique.*

Suppose $v \in C_1$. Let $C'_2, \ldots, C'_k$ the color classes of neighbors of v, that is, $C'_2 = C_2 \cap N(v), \ldots, C_k = C'_k \cap N(v)$. We can do a little better. Suppose that the color index of a node v is $k-1$. We restrict the graph G to the set of neighbors of v, that is, we consider the subgraph H of G induced by $N(v)$. Clearly, the nodes of H are legally colored using $(k-1)$ colors. Note that there must be an edge running between any two distinct color classes in H since otherwise there is no any $(k-1)$-clique in H and consequently there is no any k-clique in G. This property is valid for k-partite graphs and not valid for generic graphs.

Property 2. *If there are two color classes of neighbors of v, C'_r and C'_q, such as there is no edge between them, that is, $\forall x \in C'_r, \forall y \in C'_q, \{x, y\} \notin E$, then node v can be deleted from G without losing any k-clique.*

Definition 2. *The color index of an edge $\{u, v\}$ of G (with respect to a legal coloring of the nodes of G) is the number of color classes C_i that contain at least one node adjacent to u and v simultaneously.*

Note that if the color index of a edge $\{u, v\}$ is less than $k-2$, then $\{u, v\}$ cannot be a node of a k-clique in G. In other words:

Property 3. *If the color index of an edge $\{u, v\}$ is less than $k-2$, then the edge $\{u, v\}$ can be deleted from G when one is looking for a k-clique in G. (We do not delete the nodes u or v).*

Suppose $v \in C_1, u \in C_2$. Let $C'_3, \ldots, C'_k$ the color classes of neighbors of $\{u, v\}$, that is, $C'_2 = C_2 \cap N(v) \cap N(u), \ldots, C_k = C'_k \cap N(v) \cap N(u)$. We can do better. Let us assume that the color index of an edge $\{u, v\}$ is $k-2$. We restrict the graph G to the common neighbors of the nodes u and v. In other words, we consider the subgraph H of G induced by $N(v) \cap N(u)$. The nodes of the graph H are legally colored using $(k-2)$ colors. There must be an edge running between any two distinct color classes of H since otherwise H cannot contain a $(k-2)$-clique and so G cannot contain any k-clique. This property holds for k-partite graphs and does not hold for generic graphs.

Property 4. *If there are two color classes of neighbors of $\{u, v\}$, C'_r and C'_q, such as there is no edge between them, that is, $\forall x \in C'_r, \forall y \in C'_q, \{x, y\} \notin E$, then edge $\{u, v\}$ can be deleted from G without losing any k-clique. (We do not delete the nodes u or v).*

2.2. Dominance

Suppose that v is a node of a finite simple graph G. Clearly, if one locates a $(k-1)$-clique among the neighbors of v, then this clique can be extended to a k-clique by adding v to it. Let us assume that two nodes v and u have the same set of neighbors, that is, let us assume that $N(v) = N(u)$ holds. If one locates a $(k-1)$-clique Δ in this common neighborhood of v and u, then this clique Δ can be extended to a k-clique by adding either v or u to it. Detecting such a $(k-1)$-clique in G that can be extended to a k-clique in more than one ways opens up ways for further kernelization tricks.

Definition 3. *Let G be a graph and let a, b be distinct nodes of G. We say that node b dominates node a if a and b are not adjacent and $N(a) \subseteq N(b)$.*

The basic observation is that if among the neighbors of a dominated node, that is, in the set $N(a)$, there is a $(k-1)$-clique, then this clique can be extended to a k-clique in two ways. Namely, adding the dominated node a or alternatively adding the dominating node b to it.

Property 5. *If node a dominates node b, the dominated node b can be dropped from the graph during the search for a k-clique.*

Note that we may lose k-cliques during this reduction. But we are not going to lose all of them.

Definition 4. *Let G be a graph and let a, u, b be distinct nodes of G such that $\{a,u\}$, $\{u,b\}$ are edges of G. We say that edge $\{u,b\}$ dominates edge $\{a,u\}$ if $b \notin [N(a) \cap N(u)]$ and $[N(a) \cap N(u)] \subseteq [N(u) \cap N(b)]$.*

Property 6. *If edge $\{u,b\}$ dominates edge $\{a,u\}$, then the edge $\{a,u\}$ can be canceled from G when we are deciding if G contains a k-clique. (We do not delete the nodes a or u).*

For a proof, see [13].

Definition 5. *Let G be a graph and let x, y, u, v be distinct points of G such that $\{x,y\}$, $\{u,v\}$ are edges of G. We say that edge $\{u,v\}$ dominates edge $\{x,y\}$ if $\{u,x\}$ or $\{u,y\}$ is not edge of G, and $\{v,x\}$ or $\{v,y\}$ is not edge of G, and $[N(x) \cap N(y)] \subseteq [N(u) \cap N(v)]$.*

Property 7. *If edge $\{u,v\}$ dominates edge $\{x,y\}$, then the edge $\{x,y\}$ can be canceled from G when we are deciding if G contains a k-clique. (We do not delete the nodes x or y).*

For a proof, see [13].

The dominance relation described in Definition 4 is not a symmetric relation. We introduce now a symmetric relation motivated by the dominance relation.

Definition 6. *Let G be a finite simple graph. We say that the distinct nodes a, b of G are in β relation if a and b are not adjacent in G and a node of the set $N(a) \setminus N(b)$ is never connected to a node of the set $N(b) \setminus N(a)$ by an edge.*

Property 8. *When the nodes a, b of G are in β relation, then we construct a new graph G' from G. We delete the nodes a and b from G and add a new node c to G. We connect node c to each element in $N(a) \cup N(b)$ by an edge to get the graph G'.*

We shall call this transformation the β transformation.

Lemma 1. *Using the notations above the equation, if graph G is transformed to G' by β transformation, $\omega(G) = \omega(G')$ holds.*

Proof. We will show that $\omega(G) \leq \omega(G')$ and $\omega(G') \leq \omega(G)$.

In order to verify $\omega(G) \leq \omega(G')$ we set $k = \omega(G)$. As $\omega(G) = k$, the graph G contains a k-clique Δ.

If $a, b \notin \Delta$, then Δ is a k-clique in $\omega(G')$ and so it follows that $k \leq \omega(G')$.

If $a \in \Delta$, then $b \notin \Delta$ must hold. Here, we used the fact that nodes a and b are not adjacent in G. Note that $\Delta \cap N(a)$ is a $(k-1)$-clique Δ^* in G. Augmenting the $(k-1)$-clique Δ^* with the node c we get a k-clique Δ' in G'. Again it follows that $k \leq \omega(G')$.

If $b \in \Delta$, then $a \notin \Delta$ must hold. A similar argument we have just seen gives that $k \leq \omega(G')$.

In order to verify $\omega(G') \leq \omega(G)$ we set $k = \omega(G')$. As $\omega(G') = k$, the graph G' contains a k-clique Δ'.

If $c \notin \Delta'$, then Δ' is a k-clique Δ in G. Since G contains a k-clique, we get $k \leq \omega(G)$.

If $c \in \Delta'$, then $\Delta' \cap N(c) = \Delta' \cap [N(a) \cup N(b)]$ is a $(k-1)$-clique Δ^* in G'. Note that the nodes of Δ^* are all either in $N(a)$ or are all in $N(b)$. Here we used the fact that a node of $N(a) \setminus N(b)$ cannot be adjacent to any node of $N(b) \setminus N(a)$. Augmenting the $(k-1)$-clique Δ^* either with the node a or with the node b we get a k-clique Δ in G. It follows that $k \leq \omega(G)$. □

We would like to point out that in the particular case $N(a) \subseteq N(b)$ the condition that a node of the set $N(a) \setminus N(b)$ is never connected to a node of the set $N(b) \setminus N(a)$ by an edge plainly holds as $N(a) \setminus N(b)$ is the empty set. Therefore when node b dominates node a, then the nodes are in β relation. Note the graph G' can be constructed from G by simply deleting node a from G. Thus, the β relation provides a stronger preconditioning method than the dominance of nodes.

Let a be a vertex of a finite simple graph $G = (V, E)$. The set of edge neighbors of a consists of all the edges of G whose both end nodes are adjacent to a. The set of edge neighbors of a is denoted by $EN(a)$. In notation $EN(a) = \{\{x, y\} : x, y \in V, \{a, x\}, \{x, y\}, \{y, a\} \in E\}$.

Definition 7. *Let G be a graph and let a, b be distinct nodes of G. We say that node b dominates node a if a and b are not adjacent and $EN(a) \subseteq EN(b)$.*

Let us delete the node a from G and let us denote the new graph by G'.

Lemma 2. *$\omega(G) = \omega(G')$ holds.*

Proof. We will show that $\omega(G) \leq \omega(G')$ and $\omega(G') \leq \omega(G)$.

In order to verify $\omega(G) \leq \omega(G')$ we set $k = \omega(G)$. As $\omega(G) = k$, the graph G contains a k-clique Δ.

If $a \notin \Delta$, then Δ is a k-clique in $\omega(G')$ and so it follows that $k \leq \omega(G')$.

As a and b are not adjacent in G, $a \in \Delta$ implies $b \notin \Delta$. Note that $\Delta \cap N(a)$ is a $(k-1)$-clique Δ^* in G. Adding node b to the $(k-1)$-clique Δ^* we get a k-clique Δ' in G'. Again it follows that $k \leq \omega(G')$.

In order to verify $\omega(G') \leq \omega(G)$ we set $k = \omega(G')$. As $\omega(G') = k$, the graph G' contains a k-clique Δ'. This clique is also a clique in G. Since G contains a k-clique, we get $k \leq \omega(G)$. □

Definition 8. *Let G be a finite simple graph. We say that the distinct nodes a, b of G are in β relation if a and b are not adjacent in G and an end node of any edge in the set $EN(a) \setminus EN(b)$ is never connected to an end node of an edge in the set $EN(b) \setminus EN(a)$ by an edge.*

When the nodes a, b of G are in β relation, then we construct a new graph G' from G. We delete the nodes a and b from G and add a new node c to G. We connect node c to the end nodes of each edge in $EN(a) \cup EN(b)$ by an edge to get the graph G'.

The next result can be proved in a similar way as Lemma 1.

Lemma 3. *Using the notations above the equation $\omega(G) = \omega(G')$ holds.*

2.3. Red Black Edges

Let $G = (V, E)$ be a finite simple graph whose nodes are legally colored using k colors and suppose that we are looking for a k-clique in G. From G we construct a new graph H whose edges are colored with read and black colors. The vertex set of H is identical with the vertex set V of G. Let x and y be distinct vertices of G and consider the subgraph L of G induced by the set $N(x) \cap N(y)$. Suppose that L does not contain any $(k-2)$-clique, that is, $\omega(L) < k-2$. If the unordered pair $\{x, y\}$ is an edge of G, then we connect x and y with a red edge in H. If the unordered pair $\{x, y\}$ is not adjacent in G, then we leave x and y non-adjacent in H. Next assume that L admits a $(k-2)$-clique, that is, $\omega(L) \geq k-2$. If the unordered pair $\{x, y\}$ is an edge of G, then we connect x and y with a black edge in H. If the unordered pair $\{x, y\}$ is not adjacent in G, then we leave x and y non-adjacent in H. The definition of the coloring of the edges of the graph H is summarized in Table 1.

Table 1. Complete coloring of the edges of the auxiliary graph H associated with the given graph G.

	$\omega(L) < k-2$	$\omega(L) \geq k-2$
$\{x, y\} \in E$	$\{x, y\}$ is a red edge of H	$\{x, y\}$ is a black edge of H
$\{x, y\} \notin E$	$\{x, y\}$ is not an edge of H	$\{x, y\}$ is not an edge of H

The graphs G and H have exactly the same vertices and edges. The only difference between them is that the edges of H are colored with red and black colors. A k-clique in G corresponds to a k-clique in H. We will use the following fact that the edges of a k-clique in H are all must receive black color. We would like to use this information later, so state it formally.

Property 9. *We color the edges of H such as, that any red edge $\{x, y\}$ of graph H cannot be part of a k-clique in H.*

Note that if each edge of H is colored red, then the original graph G does not admit any k-clique. Thus, coloring the edges of H is computationally at least as demanding as computing the clique number of G. We will use a not fully completed version of the graph H. A red edge $e = \{x, y\}$ of H conveys the information that the subgraph L of G does not admit any $(k-2)$-clique. We can utilize the color indices for finding such edges like Property 3 and Property 4. While a black edge $e = \{x, y\}$ of H means that the unordered pair $\{x, y\}$ is an edge in the original graph G and we do not know whether the subgraph L contains a $(k-2)$-clique or not. This partially completed version of the graph H can be constructed with less computational effort than the completed version. Note, that this relaxed type of coloring still maintains the Property 9. The definition of the coloring of the edges of the graph H is summarized in Table 2.

Table 2. Partial coloring of the edges of the auxiliary graph H associated with the original graph G.

	We Know That $\omega(L) < k-2$	We Do Not Know If $\omega(L) \geq k-2$
$\{x, y\} \in E$	$\{x, y\}$ is a red edge of H	$\{x, y\}$ is a black edge of H
$\{x, y\} \notin E$	$\{x, y\}$ is not an edge of H	$\{x, y\}$ is not an edge of H

We can also use an extended method to construct graph H. The node set of H is still the same as the node set of G, but we will use some extra edges in H. Namely, we will add edge $\{x, y\}$ to H as a red edge if it still preserves Property 9. For this we consider the subgraph L' of H induced by the set $N(x) \cap N(y)$ in case that $\{x, y\} \notin E$. We inspect L', and if we find that it cannot contain any k-clique we connect x and y by a red edge in H. The definition of the coloring of the edges of the graph H is summarized in Table 3.

Obviously the extended graph H depends on the way we add edges to it and by following different edge adding procedures we may end up with different edge sets.

Table 3. Partial coloring of the edges of the auxiliary graph H associated with the original graph G.

	We Know That $\omega(L') < k-2$	**We Do Not Know If $\omega(L') \geq k-2$**
$\{x,y\} \in E$	$\{x,y\}$ is a red edge of H	$\{x,y\}$ is a black edge of H
$\{x,y\} \notin E$	$\{x,y\}$ is a red edge of H	$\{x,y\}$ is not an edge of H

We may talk about a reduced neighborhood of a node a in H. The set of all the nodes of H that are adjacent to node a along black edges in H is denoted by $N_b(a)$ and called the reduced neighborhood of a in H. We define color index of a node, an edge in H and we define dominance of nodes in H exploiting the red black coloring of the edges.

Suppose that the nodes of G are legally colored using k colors such that $C_1, \ldots, C_k$ are all the color classes of this coloring.

Definition 9. *The color index of a node v of H (with respect to a legal coloring of the nodes of G) is the number of color classes C_i that contain at least one node adjacent to v along an edge that is colored black in H.*

Note that if the color index of a node v is less than $k-1$, then v cannot be a node of a k-clique in G. Phrasing it differently if the colors index of v is at most $k-2$, then v can be deleted from G without losing any k-clique.

Definition 10. *The color index of a black edge $\{u,v\}$ of H (with respect to a legal coloring of the nodes of G) is the number of color classes C_i that contain at least one node adjacent to u and v simultaneously along edges receiving black color in H.*

The basic observation we need is that if the color index of an edge $\{u,v\}$ is less than $k-2$, then the edge $\{u,v\}$ can be deleted from G when one is looking for a k-clique in G. Of course we may turn the color of the edge $\{u,v\}$ from black to red in H.

Definition 11. *Let G be a graph and let H be the graph associated with G. Suppose that a, b be distinct nodes of H. We say that node b dominates node a if a and b are not adjacent in H and $N_b(a) \subseteq N(b)$.*

The content of the following lemma is that if b dominates a (by Definition 11), then the node a can be deleted from G when we are looking for a k-clique in G. Since $N_b(a) \subseteq N(a)$ clearly holds, the new dominance relation is a better preconditioning tool, than the original dominance. Let G' be the graph we get from G by deleting node a.

Lemma 4. *Using the notations above the equation $\omega(G) = \omega(G')$ holds.*

Proof. We will show that $\omega(G) \leq \omega(G')$ and $\omega(G') \leq \omega(G)$.

In order to verify $\omega(G) \leq \omega(G')$ we set $k = \omega(G)$. As $\omega(G) = k$, the graph G contains a k-clique Δ. Note that Δ is also a k-clique in H and each edge in Δ is black.

If $a \notin \Delta$, then Δ is a k-clique in $\omega(G')$ and so it follows that $k \leq \omega(G')$.

As a and b are not adjacent in G, $a \in \Delta$ implies $b \notin \Delta$. Note that $\Delta \cap N(a)$ is a $(k-1)$-clique Δ^* in G and all of its edges are black in H. Adding node b to the $(k-1)$-clique Δ^* and using the fact that $N_b(a) \subseteq N(b)$ we get a k-clique Δ' in H. An edge connecting b to a node of Δ^* receives either red or black color in H. The important aspect for us that each of these edges is an edge in both G and G' too. Therefore, we have located a k-clique Δ' in G'. It follows that $k \leq \omega(G')$.

In order to verify $\omega(G') \leq \omega(G)$ we set $k = \omega(G')$. As $\omega(G') = k$, the graph G' contains a k-clique Δ'. This clique is also a clique in G. Since G contains a k-clique, we get $k \leq \omega(G)$. □

Definition 12. *Let G be a graph and let H be the associated graph. Assume that a, u, b be distinct nodes of H such that $\{a, u\}, \{u, b\}$ are black edges of H. We say that edge $\{u, b\}$ dominates edge $\{a, u\}$ if $b \notin [N_b(a) \cap N_b(u)]$ and $[N_b(a) \cap N_b(u)] \subseteq [N_b(u) \cap N(b)]$.*

If edge $\{u, b\}$ dominates edge $\{a, u\}$, then the edge $\{a, u\}$ can be canceled from G when we are deciding if G contains a k-clique.

Definition 13. *Let G be a graph and let H be the graph we associate with G. Suppose that x, y, u, v be distinct points of H such that $\{x, y\}, \{u, v\}$ are black edges in H. We say that edge $\{u, v\}$ dominates edge $\{x, y\}$ if $\{u, x\}$ or $\{u, y\}$ is not black edge in H, and $\{v, x\}$ or $\{v, y\}$ is not black edge of H, and $[N_b(x) \cap N_b(y)] \subseteq [N(u) \cap N(v)]$.*

If edge $\{u, v\}$ dominates edge $\{x, y\}$, then the edge $\{x, y\}$ can be canceled from G when we are deciding if G contains a k-clique.

Definition 14. *Let G be a finite simple graph and let H be the associated graph. We say that the distinct nodes a, b of H are in β relation if a and b are not adjacent in H and a node of the set $N_b(a) \setminus N(b)$ is never connected to a node of the set $N_b(b) \setminus N(a)$ by a black edge in H.*

When the nodes a, b of G are in β relation, then we construct a new graph G' from G. We delete the nodes a and b from G and add a new node c to G. We connect node c to each element in $N_b(a) \cup N_b(b)$ by an edge to get the graph G'. These edges receive color black in the graph H' associated with the graph G'.

Lemma 5. *Using the notations above the equation $\omega(G) = \omega(G')$ holds.*

Proof. We construct a graph G' from H the following way. The nodes of G' are the same as nodes of G which are the same as the nodes of H. We connect nodes x, y if they are connected in H by a black edge. By the rules above G' has a k-clique if and only if G has a k-clique. Using Property 9 we may add any edge that is red to H without giving rise to a new k-clique. In other words, we allow transformation of adding such red edges that preserves Property 9. Thus, we may add the red edges mentioned in Definition 14.

The proof of the lemma can be completed using a similar argument in connection with the graph G' and evoking Definition 13. □

2.4. Edge-to-Node Transformation

Let $G = (V, E)$ be a finite simple graph. We assume that the nodes of G are legally colored using k colors and we are looking for a k-clique in G. We construct a new graph G' from G. Let C_1, C_2 be two distinct color classes of the nodes of G. Let $e_1, \ldots, e_s$ be all the edges of G such that the end nodes of the edges are all in $C_1 \cup C_2$. We delete each element of the set $C_1 \cup C_2$ from G and we add new nodes $u_1, \ldots, u_s$ to G to get G'. If $e_i = \{x_i, y_i\}$, then we connect the node u_i to each element in $N(x_i) \cap N(y_i)$ with an edge. We color the nodes of G'. The nodes in the set $V \setminus (C_1 \cup C_2)$ will inherit the colors from the coloring of the nodes of G. We assign a new color c to the new nodes $u_1, \ldots, u_s$. In this way, we get a legal coloring of the nodes of the graph G' using $k - 1$ colors.

Lemma 6. *Using the notations above the equation $\omega(G) = k$ is equivalent to the equation $\omega(G') = k - 1$.*

Proof. Suppose that $\omega(G) = k$. Let Δ be a k-clique in G. Clearly, Δ has two nodes x, y such that $x \in C_1$ and $y \in C_2$. When we constructed the graph G' from G we replaced the

edge $e = \{x, y\}$ by the node u in G'. In other words the k-clique Δ in G gives rise to a $(k-1)$-clique Δ' in G'. In general, each k-clique in G gives rise to a $(k-1)$-clique in G'. Therefore, $\omega(G) = k$ implies $\omega(G') = k-1$.

Next, suppose that $\omega(G') = k-1$. Let Δ' be a $(k-1)$-clique in G'. The clique Δ' must have a node u which is colored with the newly introduced color c. There is an edge $e = \{x, y\}$ of G which gives rise to the node u of G'. Now $[\Delta' \setminus \{u\}] \cup \{x, y\}$ is a k-clique in G. Each $(k-1)$-clique Δ' in G' gives rise to a k-clique Δ in G. Therefore $\omega(G') = k-1$ implies $\omega(G) = k$. □

The result of Lemma 6 can be utilized for preconditioning. The nodes of the graph G are legally colored with k colors and we are looking for a k-clique in G. The nodes of the constructed graph G' are legally colored with $k-1$ colors and we are looking for a $(k-1)$-clique in G'. Note that the number of nodes of the graph $G' = (V', E')$ is equal to $|V| - |C_1| - |C_2| + s$. Thus, if $s \leq |C_1| + |C_2|$, then $|V'| \leq |V|$, that is, G' has at most as many nodes as G.

Property 10. *Let us have a look at to the special case when $|C_1| = |C_2| = 2$. In this case $s \leq 4$. Consequently, the $s \leq |C_1| + |C_2|$ the condition is satisfied and so the number of nodes of G' is at most the number of nodes of G. The resulting new clique search problem is simpler than the original as we are looking for $(k-1)$-cliques in the new graph G'.*

3. Structions

We introduce some notations. Let $V = \{1, \ldots, n\}$ be the set of nodes of G. We may assume that node 1 is the pivot node, that is, the node which plays a pivotal role in the struction construction. This is only a matter of renaming the nodes of the graph G.

Let $A = \{a_1, \ldots, a_r\}$ be the set of neighbors of the pivot node 1 and let $B = \{b_1, \ldots, b_s\}$ be the set of non-neighbors of the pivot node 1. Using the set B we construct a new set C. The elements $b_1, \ldots, b_s$ of B are listed such that

$$b_1 < \cdots < b_s \tag{1}$$

holds. The ordered pair (b_α, b_β) is an element of C whenever the unordered pair $\{b_\alpha, b_\beta\}$ is an edge of G and $\alpha < \beta$. We denote such an ordered pair by $c(b_\alpha, b_\beta)$.

The set of nodes of the struction graph G' is $A \cup C$. Two distinct elements a_i and a_j from A are adjacent in G' if the unordered pair $\{a_i, a_j\}$ is an edge of G. Otherwise a_i and a_j are not adjacent in G'.

Two distinct elements $c(b_\alpha, b_\beta)$ and $c(b_\gamma, b_\delta)$ from C are adjacent in G' if $b_\alpha = b_\gamma$ and the unordered pair $\{b_\beta, b_\delta\}$ is an edge of G. Otherwise $c(b_\alpha, b_\beta)$ and $c(b_\gamma, b_\delta)$ are not adjacent in G'.

An element $c(b_\alpha, b_\beta)$ from C and an element a_i from A are adjacent in G' if the unordered pairs $\{b_\alpha, a_i\}$ and $\{b_\beta, a_i\}$ are edges of G. Otherwise the elements $c(b_\alpha, b_\beta)$ and a_i are not adjacent in G'.

Lemma 7. *If Δ' is a clique in G', then there is a clique Δ in G, where $|\Delta| = |\Delta'| + 1$.*

Proof. If Δ' contains no node from C, then $\Delta = \Delta' \cup \{1\}$ is such a required clique.

If Δ' contains nodes from C, they must be of the form $c(i, b_\alpha)$, $c(i, b_\beta)$, $c(i, b_\gamma)$, …, and the nodes $i, b_\alpha, b_\beta, b_\gamma, \ldots$ are the nodes of a clique in G. Then $\Delta = \Delta' \setminus \{c(i, b_\alpha), c(i, b_\beta), c(i, b_\gamma), \ldots\} \cup \{i, b_\alpha, b_\beta, b_\gamma, \ldots\}$ is a clique of G. □

Lemma 8. *If Δ is a clique in G, then there is a clique Δ' in G', where $|\Delta'| = |\Delta| - 1$.*

Proof. Let Δ be a clique in G, and set $\Delta_0 = \Delta \cap (A \cup \{1\})$.

If $|\Delta_0| = 0$ then removing any node from Δ results a required clique Δ' in G'.

If $|\Delta_0| = 1$ then $\Delta' = \Delta \setminus \Delta_0$ is a required clique.

If $\Delta_0 = \{i_1, i_2, \ldots, i_r\}$, where $r \geq 2$ and $i_1 < i_2 < \cdots < i_r$, then $\Delta' = (\Delta \setminus \Delta_0) \cup \{c(i_1, i_2), c(i_1, i_3), \ldots, c(i_1, i_r)\}$ is a required clique. □

The following result is a corollary to Lemmas 7 and 8.

Theorem 1. $\omega(G) - 1 = \omega(G')$.

Node Coloring and Structions

Let us suppose that the nodes of G are legally colored with the colors $1, \ldots, k$. We will use the function $f : V \to \{1, \ldots, k\}$ to describe the coloring of the nodes. Here, the equation $f(v) = i$ means that node v receives color i.

We may assume that $f(1) = 1$ holds, that is, the pivot element is colored with color 1. The reason is the following. If $f(1) = 1$ does not hold, say $f(1) = 2$ holds, then we exchange the roles of the colors 1 and 2.

We also assume that during the struction construction, the nodes of G are arranged such that the condition

$$f(1) \leq \cdots \leq f(n) \tag{2}$$

satisfied. In plain English we list first the nodes colored with color 1 and we list last the nodes receiving color k. Then we keep this list of the nodes fixed during the construction.

Lemma 9. *Let G be a finite simple graph and let G' be one of its structions. If the nodes of G can be legally colored using k colors, then the nodes of G' can be legally colored using $k - 1$ colors.*

Proof. Using the legal coloring of the nodes of G, that is, using the function f we define a coloring of the nodes of the struction graph G'. We describe the coloring by the function $g : (A \cup C) \to \{1, \ldots, k\}$. We set $g(a_i) = f(a_i)$ for each $a_i \in A$ and we set $g(c(b_\alpha, b_\beta)) = f(b_\beta)$ for each $c(b_\alpha, b_\beta) \in C$.

First we try to establish that the function g provides a legal coloring of the nodes of G'. We pick an edge of G' and try to verify that the end nodes of the edge receive distinct colors.

Let us consider an edge of type $\{a_i, a_j\}$ of G', where $a_i, a_j \in A$. Assume on the contrary that $g(a_i) = g(a_j)$ holds. The computation

$$\begin{aligned} f(a_i) &= g(a_i) \\ &= g(a_j) \\ &= f(a_j) \end{aligned}$$

leads to the contradiction that the end nodes of the edge $\{a_i, a_j\}$ of G receive the same color.

Let us consider an edge of type $\{c(b_\alpha, b_\beta), c(b_\alpha, b_\delta)\}$ of G', where $c(b_\alpha, b_\beta), c(b_\alpha, b_\delta) \in C$. Assume on the contrary that $g(c(b_\alpha, b_\beta)) = g(c(b_\alpha, b_\delta))$ holds. The computation

$$\begin{aligned} f(b_\beta) &= g(c(b_\alpha, b_\beta)) \\ &= g(c(b_\alpha, b_\delta)) \\ &= f(b_\delta) \end{aligned}$$

leads to the contradiction that the end nodes of the edge $\{b_\beta, b_\delta\}$ of G receive the same color.

Let us consider an edge of type $\{c(b_\alpha, b_\beta), a_i\}$ of G', where $c(b_\alpha, b_\beta) \in C$ and $a_i \in A$. Assume on the contrary that $g(c(b_\alpha, b_\beta)) = g(a_i)$ holds. The computation

$$\begin{aligned} f(b_\beta) &= g(c(b_\alpha, b_\beta)) \\ &= g(a_i) \\ &= f(a_i) \end{aligned}$$

leads to the contradiction that the end nodes of the edge $\{b_\beta, a_i\}$ of G receive the same color.

Next, we try to establish that the function g does not take the value 1 on. In other words we try to show that the nodes of the struction graph G' are colored with the colors $2, \ldots, k$. We pick a node of G' and try to show that the node is not colored with color 1.

Let us consider a node of type a_i, where $a_i \in A$ and assume on the contrary that $g(a_i) = 1$. Note that the unordered pair $\{1, a_i\}$ is an edge of G. (Remember the neighbors of the pivot node 1 are the elements of A.) Using $f(a_i) = g(a_i) = 1$ we end up with the contradiction that the end nodes of edge $\{1, a_i\}$ of G receive the same color.

Let us consider a node of type $c(b_\alpha, b_\beta)$ of G', where $c(b_\alpha, b_\beta) \in C$ and assume on the contrary that $g(c(b_\alpha, b_\beta)) = 1$. Note that the unordered pair $\{b_\alpha, b_\beta\}$ is an edge of G. Using $f(b_\beta) = g(c(b_\alpha, b_\beta)) = 1$ and $1 \leq f(b_\beta) \leq f(b_\alpha) = 1$ we get $f(b_\beta) = f(b_\alpha) = 1$. (Bear in mind the conditions (2) and (1)). Therefore, we end up with the contradiction that the end nodes of the edge $\{b_\alpha, b_\beta\}$ of G receive the same color. □

Lemma 10. *Using the notations introduced in the proof of Lemma 9 the node $c(b_\alpha, b_\beta)$ of the struction graph G' does not have any neighbor with color $f(b_\alpha)$.*

Proof. The color of the node $c(b_\alpha, b_\beta)$ is $g = (c(b_\alpha, b_\beta)) = f(b_\beta)$ in the graph G'. Let us assume on the contrary that the node $c(b_\alpha, b_\beta)$ has a neighbor in G' with color $f(b_\alpha)$. This neighbor is either of type a_i for some $a_i \in A$ or it is of type $c(b_\gamma, b_\delta)$ for some $c(b_\gamma, b_\delta) \in C$.

Consider an edge of type $\{c(b_\alpha, b_\beta), a_i\}$ and suppose that $f(b_\alpha)$ is the color of the node a_i in the graph G'. In notation we are assuming that $g(a_i) = f(b_\alpha)$. The unordered pair $\{c(b_\alpha, b_\beta), a_i\}$ can be an edge of G' only if the unordered pairs $\{b_\alpha, a_i\}$ and $\{b_\beta, a_i\}$ are edges of the graph G. Let us focus our attention to the colors of the end nodes of the edge $\{b_\alpha, a_i\}$ in the graph G. The color of b_α is $f(b_\alpha)$ and the color of a_i is $f(a_i)$. Now $f(a_i) = g(a_i) = f(b_\alpha)$ leads to the contradiction that the end nodes of the edge $\{b_\alpha, a_i\}$ in G receive the same color.

Consider an edge of type $\{c(b_\alpha, b_\beta), c(b_\gamma, b_\delta)\}$ and suppose that $f(b_\alpha)$ is the color of the node $c(b_\gamma, b_\delta)$ in the graph G'. In notation we are assuming that $g(c(b_\gamma, b_\delta)) = f(b_\alpha)$. The unordered pair $\{c(b_\alpha, b_\beta), c(b_\gamma, b_\delta)\}$ can be an edge of G' only if $b_\alpha = b_\gamma$ and the unordered pair $\{b_\beta, b_\delta\}$ is an edge of the graph G. By the definition of the set C, the unordered pair $\{b_\gamma, b_\delta\}$ is an edge of the graph G. Using $b_\alpha = b_\gamma$ we get that the unordered pair $\{b_\alpha, b_\delta\}$ is an edge of the graph G. The computation

$$\begin{aligned} f(b_\alpha) &= g(c(b_\gamma, b_\delta)) \\ &= f(b_\delta) \end{aligned}$$

leads to the contradiction that the end nodes of the edge $\{b_\alpha, b_\delta\}$ of G receive the same color. □

In the original (generic) struction construction, the node set of the struction graph G' is equal to $A \cup C$, where the elements of C are in the form $c(b_\alpha, b_\beta)$, where the unordered pair $\{b_\alpha, b_\beta\}$ is an edge of G and $b_\alpha < b_\beta$ holds. We propose a modification of the original struction concept where we may use a smaller set C^* in place of C. Let $b_1, \ldots, b_s$ be the elements of B and suppose that $b_1 < \cdots < b_s$ holds. Let D be the color class of b_1 in the the graph G.

Lemma 11. *Let C^* be the set of the elements in form $c(b_\alpha, b_\beta)$, where the unordered pair $\{b_\alpha, b_\beta\}$ is an edge of G and the $b_\alpha < b_\beta$, $b_\alpha \in D$ conditions hold. In the struction construction we may replace the set C by the smaller set C^*.*

Proof. Using the node set $A \cup C$ let us construct the original generic struction graph G'. By Lemma 10, a node $c(b_\alpha, b_\beta)$ of the graph G' does not have any neighbor with color $f(b_\alpha)$. Note that the color index of the node $c(b_\alpha, b_\beta)$ is at most $k - 3$. By Lemma 9, color 1 is missing from the colors. The node does not have any neighbor from its own colors class and the node cannot have any neighbor with color $f(b_\alpha)$. Therefore node $c(b_\alpha, b_\beta)$ can

be deleted from the struction graph G' when we are looking for a $(k-1)$-clique without reducing the clique number of the graph. □

In the remaining part of this section, we will show that using structions we get a stronger result as compared to Property 10. Let $G = (V, E)$ be a graph whose nodes are legally colored with k colors, and let C_1 be a color class of G such as $|C_1| = 2$. In other words, G has at least one color class containing exactly two elements. Let $C_1 = \{p, q\}$ the two nodes in this class. We choose p to be the pivot node for the struction transformation. All other nodes in the graph fall into four pair-wise disjoint sets depending on to which of the nodes p and q they are adjacent. The sets are

$$\begin{aligned} S_1 &= \{x : \{x,p\} \in E, \{x,q\} \in E\}, \\ S_2 &= \{x : \{x,p\} \in E, \{x,q\} \notin E\}, \\ S_3 &= \{x : \{x,p\} \notin E, \{x,q\} \in E\}, \\ S_4 &= \{x : \{x,p\} \notin E, \{x,q\} \notin E\}. \end{aligned}$$

Note, that the color indices of nodes in S_4 by Definition 1 are less than $k-1$ as they do not have any neighbor in C_1, so we may freely delete the elements of S_4 by Property 1, and so we may assume that the set S_4 is empty.

Corollary 1. *Using the notations above such that the struction graph $G' = (V', E')$. Then $|V'| = |V| - 2$, that is, the struction transformation reduces the number of nodes of the original graph by* 2.

Proof. Using the notations above $A = S_1 \cup S_2$. To construct C^* we need to consider the edges $\{x, q\} \in E, x \in S_3$. As the nodes in S_3 are adjacent only to q, the number of these edges exactly the same as the number of nodes in S_3, that is, $|C^*| = |S_3|$. As $V = \{p, q\} \cup S_1 \cup S_2 \cup S_3$ and $V' = A \cup C^*$ the statement follows. □

It is worth to point out that we can freely choose the pivot node as the number of nodes in G' is independent of the pivot.

Theorem 2. *Given a graph G whose nodes are legally colored by k colors and we are to decide if it contains a k-clique. If all the color classes of G are of size* 2*, then the problem can be solved in polynomial time.*

Proof. The result follows from Corollary 1 by repeatedly applying the struction transformation. □

4. Scheduling

The job sequencing problem is an optimization problem [14], that is actively researched (see for example [15–21].) Certain products are to be produced on given machines satisfying predetermined technological order. The objective is to determine the sequence of jobs in which the various products are processed on the machines in the least possible time. The well-known standard approach recasts the problem by means of a mixed integer linear program. Here, we experiment with a more combinatorial idea.

Given a positive number T we construct an integer k and an auxiliary graph G. The graph G encodes the constraints of the job sequencing problem. If the graph G contains a k-clique (a clique with k vertices), then there is a feasible job sequencing whose total completion time is at most T. If there is no k-clique present we conclude that the scheduling cannot be performed in T completion time. So, instead of an optimization problem we are dealing with a decision problem.

The reader may also notice that a maximum clique problem solver, see [22–33], can be used to solve the k-clique problem when the nodes of the graph are legally colored using k colors. Namely, if the maximum clique solver concludes that $\omega(G) < k$, then

clearly the k-clique problem does not admit any feasible solution. On the other hand, if the maximum clique solver ends up with $\omega(G) = k$, then the k-clique problem does have a feasible solution. Note, that the legal coloring of the nodes implies the inequality $\omega(G) \leq k$, so the maximum clique solver cannot find a clique greater then k. There is number of well tested algorithms to solve the maximum clique problem and so we may use any of these algorithms to solve the k-clique problem. In a typical situation, the size of the conflict graph we get from the scheduling problem is to large for the available maximum clique solvers. However, the kernelization techniques we presented in the paper may reduce the original clique problem to such extent that it is manageable by standard maximum clique solvers. Remarkably certain non-trivial scheduling instances can be completely solved by relying only on kernelization methods.

The practical utility of the clique search framework is three fold. A greedy sequencing procedure can be used to locate a feasible but typically far from optimum job sequencing. The difficulty of the clique search problem depends on the parameter T. If T is near to completion time of the greedy schedule, then the k-clique problem is relatively easy. As T gets closer and closer to the completion time of the optimum schedule the clique search problem gets computationally harder and harder. The k-clique search version of the job sequencing problem can be used to locate a not optimum but feasible solution [6,34] that may over perform the simply greedy sequencing. The k-clique search can be carried out in a parallel fashion in a relatively straight-forward manner. (See [35].) Finally, there is a technique due to D. Knuth [36], which allows to estimate the size of the search tree of a k-clique locating algorithm. So, we have an indicator of the cost of the attempted clique search.

It must be clear that the k-clique search version of the job sequencing problem will not always be a competitor of the standard mixed integer programming reformulation. Rather it is a complementary technique which maybe useful when one needs a not necessarily optimum feasible solution. On the other hand, some problems may be more tractable this way than other ways, and this can open possibilities for portfolio algorithms.

4.1. The Clique Reformulation of the Scheduling Problem

One has to make a decision which item should be scheduled to which machine at a specified time. So, we consider a triplet (u, v, w), where the number u refers to item u. The number v means that item u is assigned to machine v. Finally, the number w tells that the work on item u is started at the time point w. We consider a list of triplets that are relevant to the scheduling at hand. There are pairs of triplets that cannot appear together in any feasible schedule. Such conflicts can be recorded by constructing a conflict graph. It turns out that a specified size conflict free set of triplets defines a feasible schedule. The graph G we use is actually the complement of the conflict graph what we may call the agreement graph and instead of looking for a independent set of size k in the conflict graph we are looking for a k-clique in the agreement graph.

The nodes of the auxiliary graph G are the triplets relevant to the schedule. That is all possible job, machine and starting time combinations. Initially, we connect all the pairs of distinct nodes by an edge. Next we delete edges that connects conflicting triplets. More precisely we delete an edge if any of the following conditions meets.

1. The machines are processing a given job violating the technologically prescribed order.
2. The time distance between two jobs is not sufficient. It does not allow enough processing time for the earlier job.
3. Processing periods of the same machine on different jobs overlap.
4. Processing periods of different machines on the same job overlap.
5. Processing of a fixed job on a fixed machine occurs several times.

Set $k = (\text{number of items}) \times (\text{number of machines})$. How large clique can appear in the graph G? The answer is that the graph G may contain a k-clique. A Gantt chart which corresponds to a feasible schedule provides an k-clique in G. How many colors can be used in a legal coloring of the nodes of the graph G? The answer is that the nodes of G can be

legally colored with k colors. The nodes $(1,1,0),\ldots,(1,1,T)$ are pair-wise non-adjacent. Here, T is the make span of the schedule. (A value what we have set for the length of a schedule.) As G has $k(T+1)$ nodes it follows that the nodes of G can be legally colored with k colors.

The above considerations can be summarized in the following more formal manner. To the ordered triplet (u,v,w), which is a node of the graph G, we assign the ordered pair $[u,v]$ as a color of the node. In this way, each node of G receives exactly one color from a palette consisting of k colors and adjacent vertices never receive the same color. This coloring procedure computationally fairly efficient. In addition, we do not use any other legal coloring of the nodes of the graph G. This is the reason why in this paper we do not need any of the known vertex coloring algorithms.

The reader may note, that our formulation is quite generic. It means that the flow shop, the job shop and the open shop problem classes all can be handled by slight variants of the same approach.

We called the graph G we constructed from the given scheduling instance the auxiliary graph associated with the scheduling instance. The absence of an edge between two vertices records a conflict between the two triples corresponding to these vertices. Thus, the graph G could be called the agreement graph assigned to the scheduling problem. The complement of G could be called the conflict graph associated with the scheduling instance. (Naturally independent sets corresponds to feasible schedules in this case.) In this paper, we will refer to G as the auxiliary graph or the agreement graph assigned to the scheduling instance.

We close this subsection with collecting arguments why the clique search approach for scheduling we propose is meaningful. The clique search procedure we use does not rely on floating point arithmetic. As a consequence the computations are free of rounding errors. The same cannot be said about linear programming based solvers. The following quote from [9] well illustrates this problem. "CPLEX: IBM ILOG CPLEX Optimization Studio (CPLEX) is a state-of-the-art commercial optimization software package. We used version 12.6 and formulated Vertex Cover through integer programming. To exactly obtain the minimum vertex cover, we set mip tolerances mipgap and mip tolerances absmipgap as zero and switched on emphasis numerical. Nevertheless, CPLEX did not produce truly optimal solutions for some instances, probably because of numerical precision issues".

Computational evidence indicates that there are scheduling instances that are hard for the earlier techniques in the same time the clique reformulation leads to an optimum solution using merely the preconditioning algorithms presented in the paper. Form computational experience one can learn the following. Picking two scheduling methods M_1 and M_2 randomly there are instances that are hard for M_1 and are less hard for M_2 and conversely there are instances that are hard for M_2 and are not so hard for M_1. Shortly, typically it is not the case that a scheduling technique strictly dominates another. Putting it differently in the portfolio of various scheduling methods one must include the clique based approach. The right question is in fact how important member is the clique reformulation of the family of scheduling techniques.

There are two further considerations we should bring into the context. It may happen that we are not looking for an optimum schedule. We simply looking for a reasonable quality feasible schedule. The clique reformulation performs well in this situation. When the specified make span T is not very near to (but above) the optimum make span, then the auxiliary graph contains a large number of k-cliques and the clique search algorithm locates one relatively quickly. If the specified make span T is nor very close to (but below) the optimum make span, then the preconditioning techniques are rather efficient and they alone can prove that the auxiliary graph does not contain any k-clique. In this way, we have some assessment how good is the not optimum feasible solution.

The scheduling problems in real life setting involve constraints that the managers on the floor impose. Constraints that arise on the floor of the shop and one could not anticipate. It may happen that some of these constraints can be incorporated to the auxiliary graph framework seamlessly. This maybe an attractive feature even if the optimality of the

schedule is compromised. A feasible not necessarily optimum schedule may have some utility while an optimal but not realistic schedule may happen to be useless.

4.2. A Toy Example

In order to illustrate the previous considerations we work out a small size scheduling example in details. Three items (or jobs) are to be scheduled on two machines(figure 1). The completion times of these works are summarized in Table 4.

Table 4. Processing times on the machines in the toy scheduling example.

Item	Machine 1	Machine 2
1	1	3
2	2	1
3	3	1

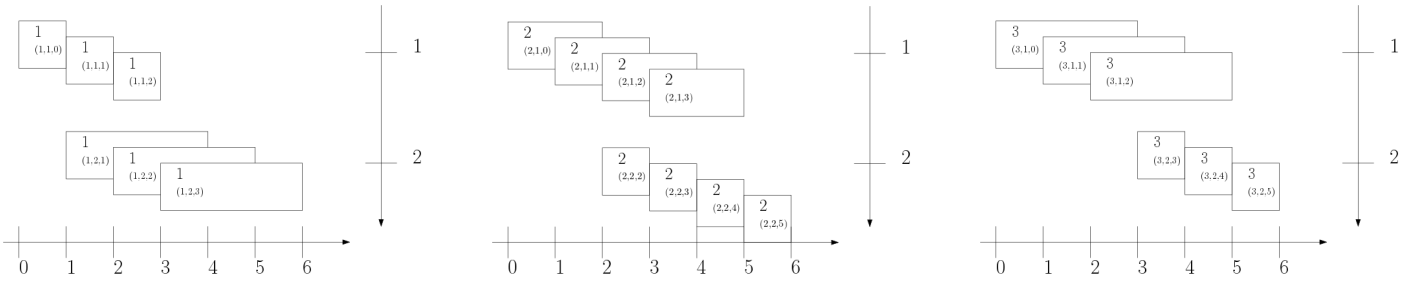

Figure 1. Possible time slots (triplets) for the given problem in Table 4 with make span of 6 h.

One can verify that there is a feasible schedule with a completion time of 7 h. We ask if there is a schedule with a completion time of 6 h. Assuming a 6 hours make span the corresponding auxiliary graph G has 20 vertices. The nodes of the graph G are triplets. We numbered the triplets by $1, \ldots, 20$, respectively. Table 5 lists the triplets together with the corresponding numbers. The adjacency matrix of the auxiliary graph G is in Figure 2. The nodes of G are legally colored using 6 colors. The color classes are the following.

$$C_1 = \{1,2,3\}, \quad C_2 = \{4,5,6\}, \quad C_3 = \{7,8,9,10\}$$
$$C_4 = \{11,12,13,14\}, \quad C_5 = \{15,16,17\}, \quad C_6 = \{18,19,20\}$$

The question is if the auxiliary graph G contains any 6-clique.

Table 5. Nodes of the auxiliary graph G in the toy scheduling example.

Name	Triplet	Name	Triplet	Name	Triplet	Name	Triplet
1	(1,1,0)	6	(1,2,3)	11	(2,2,2)	16	(3,1,1)
2	(1,1,1)	7	(2,1,0)	12	(2,2,3)	17	(3,1,2)
3	(1,1,2)	8	(2,1,1)	13	(2,2,4)	18	(3,2,3)
4	(1,2,1)	9	(2,1,2)	14	(2,2,5)	19	(3,2,4)
5	(1,2,2)	10	(2,1,3)	15	(3,1,0)	20	(3,2,5)

One may notice that nodes 2 and 3 can be removed from G as node 1 dominates them. Nodes 18 and 19 can be deleted as node 20 dominates them. The color indices of nodes $6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17$ are too small and so they maybe deleted. The graph G does not contain any 6-cliques.

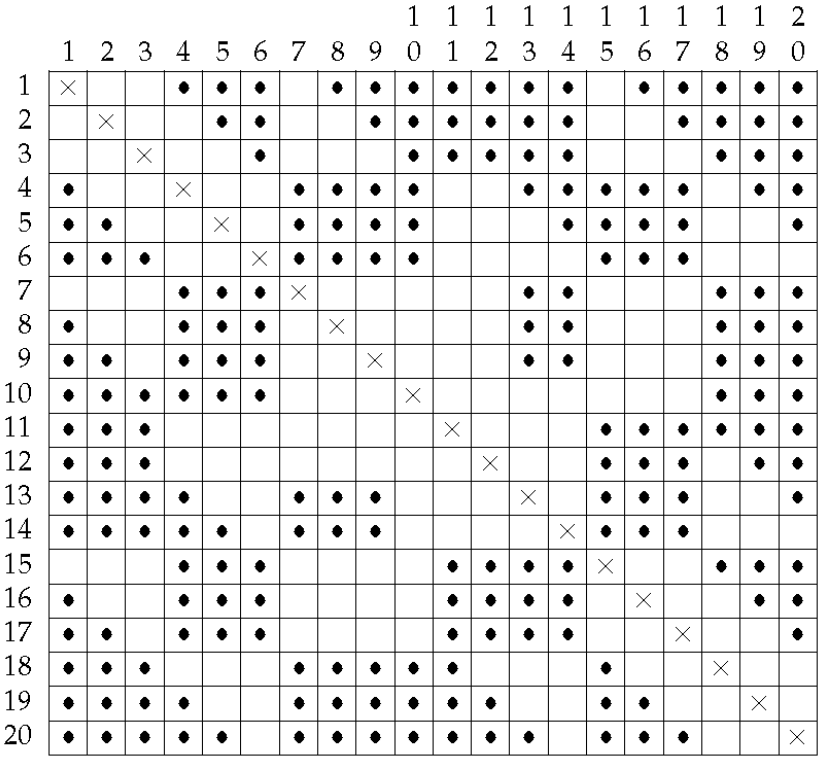

Figure 2. The adjacency matrix of the auxiliary graph G in the toy scheduling example.

5. Numerical Experiments

Given the graph reformulation of the flow shop, job shop and open shop problems we are interested in the efficiency of the clique search approach. Thus, we carried out numerical tests. First we tried to solve a non-trivial textbook example. Next we considered several large instances from the literature. As the reader will see, the clique approach can solve moderately hard problems.

In fact, for medium size problems we could use an exact k-clique solver after the kernelization, as the size of the graph was considerably reduced. For the large instances either the kernelization methods could solve the problem themselves, or the reduced graph was overly large for the exact k-clique solver. Note, that even in the latter case the number of the nodes of the graphs was usually reduced substantially.

5.1. Medium Size Flow Shop Scheduling

As an illustration of our approach, let us consider a flow shop scheduling problem where the number of jobs is 6 and the number of machines is 4. We test if there is a feasible schedule with make span $T = 385$.

The procedure for setting up the corresponding auxiliary graph gives us a graph with 5312 vertices and 11,731,403 edges. The nodes of this graph are legally colored using 24 colors and we would like to decide if the auxiliary graph contains a 24-clique.

We apply the vertex color degree kernelization, that is, we compute the color degrees of the vertices and delete the vertex if the color degree is smaller than 23. We ended up with a graph which has 5255 vertices and 11,485,115 edges.

Next, we applied the dominated node kernelization, that is, we have located dominated nodes then we deleted these nodes. The result is a graph with 4205 vertices and 7,188,467 edges.

The edges color index kernelization provided a graph with 589 vertices and 75,537 edges. An exhaustive clique search algorithm has found a 24-clique in this graph easily. In fact the number of the 24-cliques is astronomical. We have not completed the enumeration of these 24-cliques. We summarized the steps in Table 6.

Table 6. Medium size scheduling example with a make span $T = 385$.

Step	Procedure	$\|V\|$	$\|E\|$
1	setting up the auxiliary graph	5312	11,731,403
2	vertex color degree kernelization	5255	11,485,115
3	dominated node kernelization	4205	7,188,467
4	edge color index kernelization	589	75,537
5	exhausive clique search		

To proceed further we checked the dominated edge kernelization, which gave a graph with 589 nodes and 4075 edges. The exhaustive clique search algorithm this time has located altogether 13 distinct 24-cliques. The search tree had 104 nodes.

As a next step we tested if there is any feasible solution with a make span $T = 384$. The auxiliary graph corresponding to this make span has 5288 vertices and 11,622,315 edges. The nodes of the auxiliary graph are legally colored using 24 colors and we try to find a 24-clique in the graph.

The node color degree kernelization gives a graph with 5227 nodes and 11,360,283 edges. The node dominance kernelization reduces this graph to a graph with 4177 vertices and 7,090,409 edges. The edge color index kernelization deletes all edges of this graph. We summarized the steps in Table 7.

Table 7. Mediun size scheduling example with a make span $T = 384$.

Step	Procedure	$\|V\|$	$\|E\|$
1	setting up the auxiliary graph	5288	11,622,315
2	vertex color degree kernelization	5227	11,360,283
3	dominated node kernelization	4117	7,090,409
4	edge color index kernelization	0	0

We may conclude that the optimum make span of the flows shop scheduling problem we started with is $T = 385$.

5.2. Large Instances

After non-trivial medium sized instances we checked our approach for several hard problems. The resulting graphs are a magnitude larger, so we do not expect that current general exact clique solvers could solve the k-clique problem.

For testing we programmed a framework that incorporated many of the proposed kernelization methods. In each stage, we go alongside the list of these methods and try to reduce the auxiliary graph. If the reduction is big enough, that is the number deleted nodes or edges is more than a preset threshold value we start from the beginning, that is the next stage. If the threshold value is not reached in any step we stop the kernelization process. As a rule, we used the concept of red-black edges if that was possible for the kernelization in question. The time needed for each preconditioning typically increases along the list.

- If the color index of a node is less than $k - 1$, then the node can be deleted, see Property 1.
- If there is no edge between two color classes in the neighborhood of a node, then the node can be deleted, see Property 2.
- Check each pair of nodes $\{u, v\}$, delete v if u dominates v, see Property 5.
- Check each pair of nodes if β transformation can be performed on them, see Property 8.
- If there is a color class containing exactly two nodes perform struction transformation, see Corollary 1.
- If the color degree of an edge is smaller than $k - 2$, then the edge maybe deleted, see Property 3.
- Check each pair of edges $\{x, y\}, \{u, v\}$ and delete $\{x, y\}$ if $\{u, v\}$ dominates $\{x, y\}$, see Property 7.

- If there is no edge between two color classes in the common neighborhood of the two end nodes of an edge, then the edge can be removed, see Property 4.

We tested our preconditioning methods with several instances listed in [20], described originally in [15,16,37,38]. Clearly, our method at this point is capable to deal with such instances, where the number of the vertices of the auxiliary graph is at most of forty thousand. We choose five of these instances `ft06`, `abz6`, `la01`, `la04` and `orb07`. As a next step we subjected the corresponding auxiliary graphs to the preconditioning procedures.

As a detailed example we show step by step reduction of the `ft06` scheduling instance of six jobs on six machines, where the known optimum make span is 55. We set the make span to 54, and try to disprove that there is a 36-clique in the associated auxiliary graph which would correspond the six jobs and six machines. The steps of the stages are listed in Table 8. (We skipped steps where no reduction was made).

Table 8. Kernelization of the `ft06` clique search instance associated with the make span $T = 54$.

Stage	Procedure	Reduced by	$\|V\|$	$\|E\|$
0	setting up the auxiliary graph		798	275,495
1	node color index kernelization	137 nodes	661	190,801
2	node color index kernelization	6 nodes		
2	dominated node kernelization	94 nodes	561	134,411
3	dominated node kernelization	43 nodes	518	113,495
4	dominated node kernelization	21 nodes	497	104,109
5	dominated node kernelization	8 nodes		
5	if edge between node color classes	63 nodes	426	77,685
6	node color index kernelization	11 nodes	415	74,085
7	dominated node kernelization	3 nodes		
7	if edge between node color classes	17 nodes	395	67,699
8	node color index kernelization	6 nodes		
8	dominated node kernelization	4 nodes		
8	β transformation	2 nodes		
8	edge color index kernelization	18,203 edges	376	43,201
9	node color index kernelization	172 nodes	204	16,986
10	node color index kernelization	39 nodes	165	11,117
11	if edge between node color classes	123 nodes	42	545

At this point the nodes of the graph could be legally colored with less than 36 colors, so it cannot contain a 36-clique. For make span 55, the procedure reduces the auxiliary graph to a clique of the desired size, and so proves the optimality of the make span 55.

For the other four scheduling instances the kernelization proceeded as follows.

For the instance `abz6` the optimal make span value is known to be 943, and we look for a clique of size 100 in the auxiliary graph. The auxiliary graph associated with the make span 942 has 34,840 nodes. After 138 stages of kernelization, the number of the vertices of the graph was reduced to 7926 and it was possible to color the nodes legally using less then 100 colors, thus the conclusion was that it could not have a 100-clique. The auxiliary graph associated with the make span 943 has 34,940 nodes. After 145 stages of kernelization, the reduced graph turned out to be a clique. This proved the optimality of the make span 943.

For the instance `la04` the optimal make span is known to be 590. By similar steps as in the previous example the auxiliary graph was extremely reduced and no suitable clique was found for make span 589, while the clique of the appropriate size was found for make span 590. These results proved the optimality of the make span value of 590. The kernelization took 109 and 148 stages, respectively.

For the instances `la01` and `orb07` the kernelization did reduce the auxiliary graph, but the resulting graphs were still had about ten thousand vertices, so this method did not allow us to draw any definite conclusion.

6. Conclusions

The paper proposed a new type of clique search problem in which the nodes of a given graph are legally colored using k colors and we are to decide if the graph contains a k-clique.

On the one hand, we argued that there are important problems in applied discrete optimization that can be naturally modeled in this frame work. We have pointed out that in particular the job shop problem can be aptly formulated as a k-clique problem in a suitably constructed k-partite graph. In future work we intend to present further problems as well. The legal coloring of the nodes of the graph with k colors was accomplished without any extra computational effort. We do not need any of the standard vertex coloring algorithms in this situation. We were able to solve some non-trivial size scheduling instances to optimality.

On the other hand, we described a host of kernelization methods. Some of these are already known from the literature while others are new. We tailored earlier methods to the new situation when the underlying graph is k-partite. These attempts were mainly motivated by computational efficiency. The modified preconditioning procedures are always less time consuming than the original ones. For instance, when one determines the color degree of a node v in the case of a k-partite graph one may terminates the computation after detecting a color class in which v does not have any neighbor. While such action is not justified in the case of a generic graph. Further ideas coming from kernelization may lead to new theoretical insights. The best illustration we can mention is the revised struction transformation and the theorem about graphs in which each color class contains exactly two elements, Theorem 2. We also worked out several modifications to previously known kernelization methods, such as β transformation and the use of the red-black edges concept.

The theory and practice of scheduling is a well developed subject. The main question is of course that what are the merits of the clique approach of the scheduling problem.

Firstly, there are scheduling instances when the clique approach works very well compared to other methods. There are solved numerical instances and there are theoretical results demonstrating this fact. Secondly, the clique search setting avoids the numerical instabilities associated with accumulating of rounding errors in floating point computations. Thirdly, designing parallel scheduling algorithms is relatively straight-forward in the clique search setting. (See [35]) Fourthly, establishing lower and upper bounds for the optimal make span is again relatively simple when the optimum solution is out of reach. This is true in particular when the lower and upper bounds are not too tight. The clique search approach can come handy in locating a not necessarily optimal but feasible schedule. In addition, we may meet with such constraints in the scheduling problems that cannot be easily incorporated into a particular modeling framework while the constraints fit well into the auxiliary graph approach.

Author Contributions: Writing (review and editing), S.S. and B.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by National Research, Development and Innovation Office—NKFIH Fund No. SNN-135643.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Corrádi, K.; Szabó, S. A combinatorial approach for Keller's conjecture. *Period. Math. Hung.* **1990**, *21*, 95–100. [CrossRef]
2. Szabó, S.; Zavalnij, B. Reducing Graph Coloring to Clique Search. *Asia Pac. J. Math.* **2016**, *3*, 64–85.
3. Szabó, S.; Zavalnij, B. Reducing hypergraph coloring to clique search. *Discret. Appl. Math.* **2019**, *264*, 196–207. [CrossRef]
4. Depolli, M.; Szabó, S.; Zavalnij, B. An Improved Maximum Common Induced Subgraph Solver. *MATCH Commun. Math. Comput. Chem.* **2020**, *84*, 7–28.

5. Zavalnij, B. The k-Clique Problem—Usage, Modeling Expressivity, Serial and Massively Parallel Algorithms. Ph.D. Thesis, University of Szeged, Szeged, Hungary, 2020.
6. Bomze, I.M.; Budinich, M.; Pardalos, P.M.; Pelillo, M. The Maximum Clique Problem. In *Handbook of Combinatorial Optimization*; Kluwer Academic Publisher: Amsterdam, The Netherlands; New York, NY, USA, 1999; Volume 4.
7. Alexe, G.; Hammer, P.L.; Lozin, V.V.; de Werra, D. Struction revisited. *Discret. Appl. Math.* **2003**, *132*, 27–46. [CrossRef]
8. Ebenegger, C.; Hammer, P.L.; de Werra, D. Pseudo-Boolean Functions and Stability of Graphs. *N.-Holl. Math. Stud.* **1984**, *95*, 83–97.
9. Akiba, T.; Iwata, Y. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.* **2016**, *609*, 211–225. [CrossRef]
10. Chang, L.; Li, W.; Zhang, W. Computing a near-maximum independent set in linear time by reducing-peeling. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17), Chicago, IL, USA, 14–19 May 2017; pp. 1181–1196.
11. Hespe, D.; Schulz, C.; Strash, D. Scalable kernelization for maximum independent sets. In Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX), New Orleans, LA, USA, 7–8 January 2018; pp. 223–237.
12. Strash, D. On the power of simple reductions for the maximum independent set problem. In *Computing and Combinatorics (COCOON'16)*; Dinh, T.N., Thai, M.T., Eds.; Springer: Cham, Switzerland, 2016; Volume 9797, pp. 345–356.
13. Szabó, S. Parallel algorithms for finding cliques in a graph. *J. Phys. Conf. Ser.* **2011**, *268*, 012030. [CrossRef]
14. Jain, A.S.; Meeran, S. Deterministic job-shop scheduling: Past, present and future. *Eur. J. Oper. Res.* **1999**, *113*, 390–434. [CrossRef]
15. Adams, J.; Balas, E.; Zawack, D. The shifting bottleneck procedure for job shop scheduling. *Manag. Sci.* **1988**, *34*, 391–401. [CrossRef]
16. Applegate, D.; Cook, W. A computational study of job-shop scheduling. *ORSA J. Comput.* **1991**, *3*, 149–156. [CrossRef]
17. Brinkkötter, W.; Brucker, P. Solving open benchmark instances for the job-shop problem by parallel head–tail adjustments. *J. Sched.* **2001**, *4*, 53–64. [CrossRef]
18. Gharbi, A.; Labidi, M. Extending the Single Machine-Based Relaxation Scheme for the Job Shop Scheduling Problem. *Electron. Notes Discret. Math.* **2010**, *36*, 1057–1064. [CrossRef]
19. Koshimura, M.; Nabeshima, H.; Fujita, H.; Hasegawa, R. Solving Open Job-Shop Scheduling Problems by SAT Encoding. *IEICE Trans. Inf. Syst.* **2010**, *E93*, 2316–2318. [CrossRef]
20. Vilím, P.; Laborie, P.; Shaw, P. Failure-Directed Search for Constraint-Based Scheduling. In *Integration of AI and OR Techniques in Constraint Programming, CPAIOR 2015*; Lecture Notes in Computer Science; Michel, L., Ed.; Springer: Cham, Switzerland, 2015; Volume 9075.
21. Vilím, P.; Laborie, P.; Shaw, P. Failure-Directed Search for Constraint-Based Scheduling—Detailed Experimental Results. Available online: http://vilim.eu/petr/cpaior2015-results.pdf (accessed on 18 December 2021).
22. Carraghan, R.; Pardalos, P.M. An exact algorithm for the maximum clique problem. *Oper. Res. Lett.* **1990**, *9*, 375–382. [CrossRef]
23. Hasselberg, J.; Pardalos, P.M.; Vairaktarakis, G. Test case generators and computational results for the maximum clique problem. *J. Glob. Optim.* **1993**, *3*, 463–482. [CrossRef]
24. Konc, J.; Janezic, D. An improved branch and bound algorithm for the maximum clique problem. *MATCH Commun. Math. Comput. Chem.* **2007**, *58*, 569–590.
25. Li, C.-M.; Quan, Z. An Efficient Branch-and-Bound Algorithm Based on MaxSAT for the Maximum Clique Problem. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), Atlanta, GA, USA, 11–15 July 2010; pp. 128–133.
26. Li, C.-M.; Fang, Z.; Xu, K. Combining MaxSAT Reasoning and Incremental Upper Bound for the Maximum Clique Problem. In Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI2013), Herndon, VA, USA, 4–6 November 2013; pp. 939–946.
27. Li, C.-M.; Jiang, H.; Manya, F. On Minimization of the Number of Branches in Branch-and-Bound Algorithms for the Maximum Clique Problem. *Comput. Oper. Res.* **2017**, *84*, 1–15. [CrossRef]
28. Nikolaev, A.; Batsyn, M.; San Segundo, P. Reusing the Same Coloring in the Child Nodes of the Search Tree for the Maximum Clique Problem. *Learn. Intell. Optim.* **2015**, *8994*, 275–280.
29. Östergård, P.R.J. A fast algorithm for the maximum clique problem. *Discret. Appl. Math.* **2002**, *120*, 197–207. [CrossRef]
30. San Segundo, P.; Nikolaev, A.; Batsyn, M. Infra-chromatic bound for exact maximum clique search. *Comput. Oper. Res.* **2015**, *64*, 293–303. [CrossRef]
31. Szabó, S.; Zavalnij, B. A different approach to maximum clique search. In Proceedings of the 2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 20–23 September 2018; pp. 310–316.
32. Tomita, E.; Seki, T. An Efficient Branch-and-Bound Algorithm for Finding a Maximum Clique. In *Discrete Mathematics and Theoretical Computer Science*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; pp. 278–289.
33. Wu, Q.; Hao, J.-K. A review on algorithms for maximum clique problems. *Eur. J. Oper. Res.* **2015**, *242*, 693–709. [CrossRef]
34. Lamm, S.; Sanders, P.; Schulz, C.; Strash, D.; Werneck, R.F. Finding Near-Optimal Independent Sets at Scale. In Proceedings of the 16th Meeting on Algorithm Engineering and Experimentation (ALENEX'16), San Diego, CA, USA, 7–8 January 2019.
35. Szabó, S.; Zavalnij, B. Decomposing clique search problems into smaller instances based on node and edge colorings. *Discret. Appl. Math.* **2018**, *242*, 118–129. [CrossRef]
36. Knuth, D.E. Estimating the efficiency of backtrack programs. *Math. Comput.* **1975**, *29*, 121–136. [CrossRef]

37. Fisher, H.; Thompson, G.L. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling*; Muth, J.F., Thompson, G.L., Eds.; Prentice Hall: Hoboken, NJ, USA, 1963; pp. 225–251.
38. Lawrence, S. *Resource Constrained Project Scheduling. An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*; Carnegie-Mellon University: Pittsburgh, PA, USA, 1984.