Algebra and Discrete Mathematics RESEARCH ARTICLE
Volume **20** (2015). Number 1, pp. 89–114
© Journal "Algebra and Discrete Mathematics"

A tabu search approach to the jump number problem

Przemysław Krysztowiak and Maciej M. Sysło

Communicated by D. Simson

Abstract. We consider algorithmics for the jump number problem, which is to generate a linear extension of a given poset, minimizing the number of incomparable adjacent pairs. Since this problem is NP-hard on interval orders and open on two-dimensional posets, approximation algorithms or fast exact algorithms are in demand.

In this paper, succeeding from the work of the second named author on semi-strongly greedy linear extensions, we develop a metaheuristic algorithm to approximate the jump number with the tabu search paradigm. To benchmark the proposed procedure, we infer from the previous work of Mitas [Order 8 (1991), 115–132] a new fast exact algorithm for the case of interval orders, and from the results of Ceroi [Order 20 (2003), 1–11] a lower bound for the jump number of two-dimensional posets. Moreover, by other techniques we prove an approximation ratio of $n/\log\log n$ for 2D orders.

1. Introduction

The jump number problem is to find a linear extension of a given poset minimizing the number of jumps, that is, incomparable adjacent pairs. It is best motivated by the following scheduling problem. Suppose a set of jobs is to be performed by a single machine, one at a time, with respect to

2010 MSC: 90C27, 90C59.

Key words and phrases: graph theory, poset, jump number, combinatorial optimization, tabu search.

some technological precedence constraints. Every job processed after one which is not constrained to precede it requires a warm-up (here, called a jump), which leads to a fixed unit of additional cost. The objective is to find a schedule which minimizes the number of warm-ups (jumps).

A purely theoretical interest in this problem is associated with a fact proved by Habib [8] that posets with isomorphic undirected comparability graphs have equal jump number. Consequently, the jump number fits the framework of comparability invariants, together with order dimension, the number of all linear extensions, the path partition number, and other properties. This leads to characterisation questions of comparability graphs satisfying given property and of possible interpretations of these invariants.

Another related topic is a classification of jump-critical posets. We recall from [26] that P is jump-critical if for any $p \in P$, $P \backslash \{p\}$ has less jumps than P. Some results have been established by El-Zahar et al. [26–28].

The main results of this work are as follows.

1) We design and benchmark a tabu search algorithm to approximate the jump number, see Section 3 and 4. It is built upon semi-strongly greedy linear extensions, defined by the second named author in terms of arc diagram representations of posets.
2) We give in Section 4.1 a new exact algorithm for the jump number of interval orders, based on the previous work of Mitas [16].
3) We show in Section 4.3 that the jump number has an $(n/\log\log n)$ approximation ratio on two-dimensional posets.

We now outline the paper structure.

The proposed tabu search algorithm explores semi-strongly greedy linear extensions, defined by the second named author (see Section 2.2). After reviewing the respective exact algorithm in Section 2, we verify how many solutions are generated when it is applied to various posets.

Our adaptation of the tabu search paradigm is proposed in Section 3 and tested in Section 4. In benchmarks we focus on two non-trivial classes of posets: interval orders and two-dimensional orders. A previous work of Mitas [16] on interval orders contains a characterization of optimal solutions in terms of subgraph packings.

In Section 4.1 we explore this idea to obtain a new exact algorithm for these orders. This allows us to calculate the jump number in reasonable time for posets having up to several hundred elements. In effect, we obtain a testbed to verify the quality of solutions generated with the proposed tabu search algorithm.

Perhaps even more interesting is the case of two-dimensional orders, since the complexity status of the jump number problem has remained open in this class for several years now.

In Section 4.2 we exploit an interpretation given by Ceroi [4]. Chains to form a linear extension are seen as rectangles in the plane, and the bump number (see Section 1.1) corresponds to the maximum weight of an independent set (MWIS in short) of rectangles. Thus, a linear programming relaxation of the MWIS integer formulation yields a bound on $s(P)$. Even though the approximation ratio of our tabu search algorithm is unknown and only verified experimentally, we prove in Section 4.3 using other techniques that the jump number admits an $(n/\log\log n)$ approximation ratio on two-dimensional posets.

1.1. Preliminaries

We denote by $(P, <_P)$, or simply by P, a finite strict partially ordered set, in short a *poset* of cardinality$|P| = n$. That is, $<_P$ is a transitive and irreflexive relation on P. For any $p \in P$, $Succ_P(p) = \{q \in P : p <_P q\}$ is the *set of successors* of p and $Pred_P(p) = \{q \in P : q <_P p\}$ is the *set of predecessors* of p. $\mathcal{S}_P = \{Succ_P(p) : p \in P\}$ is the *family of distinct successor sets* and $\mathcal{P}_P = \{Pred_P(p) : p \in P\}$ is the *family of distinct predecessor sets* of a poset P. We say that p is covered by q if $p <_P q$ and for no r, $p <_P r <_P q$.

A *linear extension* $L = p_1, p_2, \ldots, p_n$ is a total ordering of P preserving the relation, that is, $p_i <_P p_j$ implies $i < j$. Two adjacent elements p_i, p_{i+1} in L form a *jump* if $p_i \not<_P p_{i+1}$ and otherwise they form a *bump*. Since jumps split L into chains of P, we can write $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$.

Problem 1.1.1. Let $s_L(P)$ denote the number of jumps in a linear extension L of a poset P. The *jump number problem* is to find

$$s(P) = \min\{s_L(P) : L \text{ is a linear extension of } P\}.$$

Problem 1.1.1 is equivalent to maximizing the number of bumps $b_L(P)$ amongst linear extensions of P, as for any L we have $s_L(P)+b_L(P) = n-1$. We write $b(P)$ for the maximum number of bumps in a linear extension of P. If $s_L(P) = s(P) = n - 1 - b_L(P)$ then L is called an *optimal linear extension* of P.

Definition 1.1.2. A poset $(P, <_P)$ is an *interval order* (or an *interval poset*) if there is a bijection between its elements and closed intervals on the real line, $P \longleftrightarrow \{I_p = [l(p), r(p)], l(p) \leqslant r(p)\}_{p \in P}$, such that $p <_P q$ if and only if $r(p) < l(q)$. $(P, <_P)$ is *two-dimensional* (or 2D) if $<_P$ is an intersection of two linear orders $\{L_1, L_2\}$, called a *realizer* of P.

Interval orders are characterized as posets excluding a subposet consisting of two independent 2-chains [6]. The recognition of two-dimensional posets can be accomplished in $\mathcal{O}(n^2)$ time, see [20] and also [10].

Definition 1.1.3. A chain C in P is *greedy* if $Pred(p) \cup \{p\} = C$, where $p = \sup C$, and for no element q covering p, the chain $C \cup \{q\}$ has this property. A linear extension $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$ is *greedy* if C_i is a greedy chain in $P \setminus \cup_{j<i} C_j$.

It is easy to prove that every poset has an optimal linear extension which is greedy.

1.2. Previous work

It has been proved by Pulleyblank [18] that the jump number problem is NP-hard on bipartite orders, that is, on posets having only minimal and maximal elements. Another NP-hardness proof has been given by Bouchitté and Habib [3]. Moreover, by Mitas [16] the problem remains NP-hard on interval orders. There are polynomial-time algorithms for some restricted classes of posets. These include semi-orders (i.e., interval orders formed by intervals of the same length) [2], and N-free orders (that is, with the N subposet forbidden) [19, 22]. The problem remains open on two-dimensional orders. However, Ceroi [4] proved NP-hardness of a generalized variant in which non-negative weights are associated with comparabilities and the objective is to maximize their sum on bumps of a linear extension. Due to high complexity of the problem, approximate algorithms are in demand. Those have been found only for interval orders (Sysło [25], Felsner [5], Mitas [16]). An exact algorithm has been designed by Sysło [24] (it is shortly reviewed in subsequent sections). As far as we know, the only metaheuristic approach published so far is that of Ngom [17], who adapted the genetic algorithm to the jump number problem.

2. Semi-strongly greedy linear extensions

In this paper a tabu search procedure is presented to search for valuable solutions amongst very particular greedy linear extensions, defined by the second named author. Therefore, we now quickly recall what semi-strongly greedy linear extensions are and how they are generated. These linear extensions are formed from special greedy chains, defined by means of a digraph representation of $(P, <_P)$ explained below.

2.1. Arc diagrams

A *digraph* is denoted by $D = (V, A, t, h)$, where V is the vertex set, A is the arc set, and $t, h : A \to V$ are incidence mappings (t for tail, h for head). Then each arc $a \in A$ is of the form $a = (t(a), h(a))$. A sequence of arcs $\pi = (a_1, a_2, \ldots, a_l), l \geqslant 1$ is a *path* in D if $h(a_i) = t(a_{i+1})$ for $i = 1, 2, \ldots, l-1$. By $\mathrm{tc}(D) = (V, \mathrm{tc}(A), t^\star, h^\star)$ we denote a *transitive closure* of D, where $(a_1, \ldots, a_l), l \geqslant 1$ is a path in D if and only if $\mathrm{tc}(A)$ contains an arc b such that $t^\star(a_1) = t^\star(b)$ and $h^\star(a_l) = h^\star(b)$.

Definition 2.1.1. An *arc diagram* for a poset $(P, <_P)$ is an acyclic digraph $D(P) = (V, R, t, h)$ for which there is a mapping $\phi : P \to R$ such that for every $p, q \in P, p \neq q$, we have $p <_P q$ iff $(h^\star(\phi(p)), t^\star(\phi(q))) \in R^\star$, where $t^\star, h^\star$ are the incidence mappings of $\mathrm{tc}(D)$ and $R^\star = \mathrm{tc}(R) \cup \{(v, v) : v \in V\}$. An arc $a \in \phi(P)$ is a *poset arc* and otherwise a is a *dummy arc.*

Informally, certain arcs represent the elements of P, and the purpose of the remaining ones is to preserve the comparabilities along the paths of $D(P)$.

An example is shown in Figure 1. Elements $2, 3, 12, 8$ form one of the chains in this poset, so the four corresponding arcs are aligned in one of the paths leading from the source to the sink of the diagram. In any linear extension, these (and other) order constraints have to be respected, e.g., $(2, 3, 10) \oplus (7) \oplus (14, 5, 1) \oplus (4, 13) \oplus (6, 12) \oplus (11, 9, 8)$.

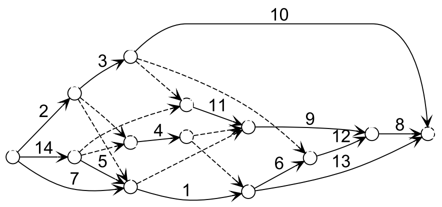

Figure 1. Arc diagram representation of a two-dimensional poset

Figure 2 is another example, whose one of linear extensions is $(1, 3) \oplus (2, 7) \oplus (4, 9) \oplus (5, 10) \oplus (6, 11) \oplus (8)$.

An algorithm to construct an adequate arc diagram for any finite poset P was given by Sysło [21]. For the sake of completeness, it is repeated as Algorithm 1.

Algorithm 1 Arc diagram for a poset P (see [21])

```
Input: A finite poset $(P, <_P)$.
Output: $D(P)$, an arc diagram for $P$.
Step 1. Let $\mathcal{P}_P = \{Pred_1, \ldots, Pred_k\}$, $\mathcal{S}_P = \{Succ_1, \ldots, Succ_l\}$.
    For each $Pred_i$ let $U_i = \bigcap_{p \in Pred_i} Succ_P(p)$.
Step 2. {The vertices:}
    Let $x_1, x_2, \ldots, x_h$ correspond to those $Pred_i$, for which there exists
        $Succ_j = U_i$.
    Let $y_{h+1}, y_{h+2}, \ldots, y_k$ correspond to remaining $Pred_i$.
    Let $z_{h+1}, z_{h+2}, \ldots, z_l$ correspond to remaining $Succ_j$.
    Let $y_i = z_i = x_i$ for $i = 1, 2, \ldots, h$.
Step 3. {The arcs:}
    For each $p \in P$ add a poset arc $(y_i, z_j)$, where
        $P_i = Pred_P(p)$, $Succ_j = Succ_P(p)$.
    For every $p, q \in P$, $p \neq q$,
        if $p$ is covered by $q$, and $z_j \neq y_i$, where
            $Succ_j = Succ_P(p)$ and $Pred_i = Pred_P(q)$,
        then add a dummy arc $(z_j, y_i)$.
    Finally, remove transitive arcs provided that they are not poset arcs.
```

2.2. Greedy paths

Definition 2.2.1. In an arc diagram $D(P)$, a natural counterpart of a greedy chain C (see Section 1.1) is a *greedy path* $\pi(C) = (a_1, a_2, \ldots, a_l)$, $l \geqslant 1$, i.e., a path satisfying the following conditions:

- No vertex of $\pi(C)$ except $h(a_l)$ is a head of any arc other than a_j, $j = 0, 1, \ldots, l-1$.
- a_j is a poset arc, $j = 1 \ldots l$.
- $\pi(C)$ cannot be extended to a longer path satisfying the above two conditions.

In the reverse direction, any greedy path π induces a greedy chain C_π in $D(P)$. Thus, an algorithm to generate a greedy linear extension can be formulated in terms of an arc diagram for P, see Algorithm 2.

Algorithm 2 Greedy linear extension

```
Input: $D(P)$, an arc diagram for $P$.
Output: $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$, a linear extension of $P$.
Step 1. { Initialization }
  $D := D(P)$
  $L := \varnothing$
Step 2. while $D \neq \varnothing$
    $(\star)$ find a greedy path $\pi$ in $D$
    $L := L \oplus C_\pi$
    $D := D(P \backslash L)$, an arc diagram for the remaining poset
Step 3. return $L$
```

It is easy to design an analogous procedure to enumerate all greedy linear extensions, but initial experiments reveal quickly that it is a very time-consuming and hence inefficient process.

However, it was proved in a series of papers [21–25], that the search space can be significantly reduced, since for every poset the class of greedy linear extensions can be further restricted to the class of very particular greedy linear extensions which contains an optimal solution. These linear extensions are composed from two special types of greedy chains, as described below.

Definition 2.2.2. A *strongly greedy path* π is a greedy path satisfying:

- either $h(\pi)$ is the sink of $D(P)$, or
- $h(\pi)$ is the head of a poset arc $b \neq a_l$ such that no path terminating with b has a vertex incident with a dummy arc.

If $D(P)$ contains at least one strongly-greedy path π then there always exists an optimal linear extension beginning with C_π. If there are no strongly-greedy paths in $D(P)$ then there is at least one *semi-strongly greedy path*, i.e., a greedy path π such that

- π has a vertex which is a tail of a dummy arc but not a head of a dummy arc.

In such case, when searching for an optimal linear extension, we have to consider all semi-strongly greedy paths. It should be noted however that not every greedy path is semi-strongly greedy, so all in all, the search space is greatly reduced in comparison with an enumeration of all greedy linear extensions.

The arc diagram in Figure 2 has three semi-strongly greedy paths $(1, 3)$, $(1, 4)$ and $(1, 5)$. The path (2) is greedy, but it is neither strongly

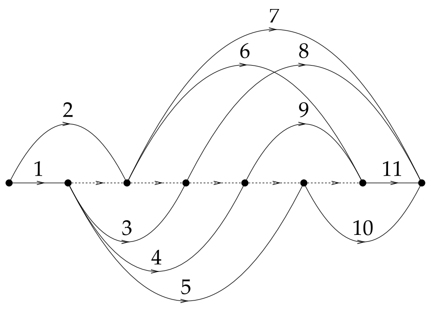

FIGURE 2. Arc diagram representation of an interval order

greedy, nor semi-strongly greedy. But the diagram in Figure 1 has two strongly greedy paths. The path $(2, 3, 10)$ passes through a tail of a dummy arc, so it would classify as semi-strongly greedy, but it also terminates in the sink, so it is strongly greedy. In addition, the vertex terminating $(14, 5)$ terminates also (7), which has no vertex incident with a dummy arc. So $(14, 5)$ is strongly greedy.

In conclusion, we have the following Theorem 2.2.3.

Theorem 2.2.3 (Sysło [23]). *Every poset has an optimal linear extension $L = C_0 \oplus C_1 \oplus \ldots \oplus C_m$, called semi-strongly greedy, such that each chain C_i is strongly greedy in $P_i = P \setminus \bigcup_{j<i} C_j$ or semi-strongly greedy in P_i if P_i has no strongly gredy chains.*

To design an algorithm generating one semi-strongly greedy linear extension, we simply replace Step **2**. $(\star)$ of Algorithm 2 with

> $(\star)$ find a strongly greedy path π in D; **if** no such path has been found **then** set π to any semi-strongly greedy path in D.

It is now also easy to devise an exact algorithm for the jump number problem, which searches for optimal solution amongst all semi-strongly greedy linear extensions via backtracking. For this purpose, in Step **2**. $(\star)$ of Algorithm 2, if there are no strongly greedy paths, then instead of choosing an arbitrary semi-strongly greedy path we verify every one of them, and apply the procedure recursively on every respective subposet. We refer to this algorithm as `OptLinExt` [24] in subsequent sections, where a new tabu search algorithm is proposed, based on these special linear extensions. For a more in-depth treatment of the topic we refer the reader to the articles of Sysło [21–25].

2.3. The running time of OptLinExt

Let k denote the number of dummy arcs in $D(P)$. It was concluded in [24] that the pessimistic time complexity of `OptLinExt` is of order $\mathcal{O}(k! \cdot \mathrm{poly}(n,k))$, since there are always at most $k!$ semi-strongly greedy linear extensions. That is, k is an important factor contributing to the complexity of the problem.

We have performed an experiment to learn how many solutions are generated in reality on posets for varying number of dummy arcs. For a fixed poset size ($n = 120$ in the case of interval orders and $n = 30$ in the case of two-dimensional orders), and for each number of dummy arcs $k \in \{5, 10, \ldots\}$, a hundred of posets were randomized, having these requested properties. To obtain posets with a given number of dummy arcs in their arc diagrams, we use a genetic algorithm, with distance from k in question being the optimality factor. The maximum number of generated solutions amongst a group of posets with k dummy arcs was recorded. This is plotted in Figure 3.

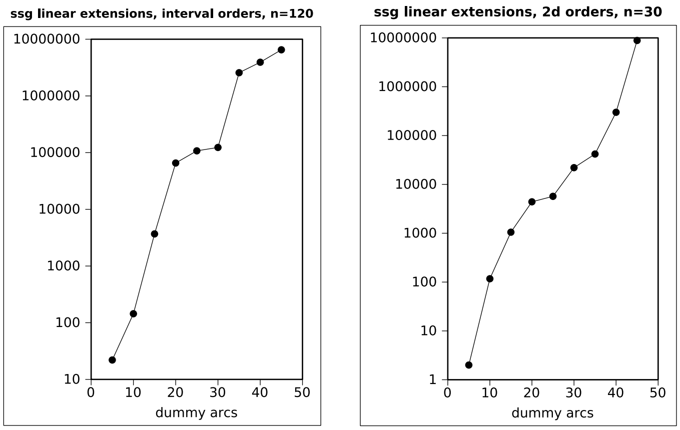

Figure 3. Total number of semi-strongly greedy linear extensions in sample posets containing increasing number of dummy arcs

This series of trials helps to asses the magnitude of the search space, browsed through by the tabu search algorithm described in Section 3. Interestingly, it shows that the search space is typically greater in the case of two-dimensional posets.

In the group of interval orders on 120 elements, containing 45 dummy arcs, a poset was spotted having 6 510 338 semi-strongly greedy linear extensions, and the exact algorithm took over 5 hours to list them. Typically, the total number of solutions is lower. On the other hand, a 30-element two-dimensional poset was found, with 45 dummy arcs in its diagram, for which the search space contains 8 857 068 semi-strongly greedy linear extensions. In this case, the execution of `OptLinExt` took 3 445 seconds, benefitting from shorter time required to rebuild arc diagrams for no more than 30 elements.

We note that the number of dummy arcs is lesser than n on interval orders. This is not the case on two-dimensional posets. Thus, it is common for two-dimensional orders to have a significantly greater search space of semi-strongly greedy linear extensions than interval orders.

3. A tabu search algorithm to approximate the jump number

Tabu search is a famous algorithmic technique of moving stepwise towards an optimal solution of a computational problem. Its characteristic feature is maintaining a list of moves not allowed at given iteration, called a tabu list. The purpose of this list is the avoidance of repeatedly visiting the same solutions. In recent years, tabu search has become a major metaheuristic paradigm to approximate hard optimization problems. The rationale behind this method can be found in the monograph of Glover and Laguna [7].

In a tabu search algorithm, every solution is treated as a point in the search space. Initially, there is some solution sol, and in each step we move to another solution sol' selected from a neighbourhood of sol. That is, a fixed number of solutions is generated from sol, and the algorithm follows to the best of them, provided it is not a tabu move. We now turn to a description of our adaptation for the jump number problem.

3.1. An adaptation for the jump number problem

In our approach, every solution is some semi-strongly greedy linear extension L of P. A neighbour solution is generated from L by splitting it between some consecutive chains, and completing it according to the semi-strongly greedy algorithm (see Section 2.2). A linear extension L is represented as a list of chains, which in turn are lists of poset elements. So if we decide to split L after kc chains, then $L[0] \oplus \ldots \oplus L[kc-1]$ becomes the initial part of a neighbour L'. Our adaptation is shown as Algorithms 3 and 4.

Algorithm 3 `TS-CompleteLinExt`

```
Input: A poset $P$, initial chains of some greedy linear extension $partialL = C_0 \oplus \ldots \oplus C_{c-1}$, and the minimum number of jumps $s_\star$ found so far.
Output: $L = C_0 \oplus \ldots \oplus C_{c-1} \oplus C_c \oplus \ldots \oplus C_m$, a linear extension of $P$.
Step 1. { Initialization }
  $D := D(P \backslash partialL)$, an arc diagram for $P \backslash partialL$
  $c :=$ the number of chains in $partialL$
  $e :=$ the number of poset elements in $partialL$
  $s_{LB} := s(partialL) + 1 + s_{LB}(D)$
  { $s_{LB}(D)$ is the lower bound on $s(P \backslash partialL)$, Thm. 3.1.1 }
Step 2. if $s_{LB} \geqslant s_\star$ then add $(c, e)$ to $TabuPositions$, return $\varnothing$
  else go to 3
Step 3. if $D$ has no more dummies than $MaxDummies$ then
    $remainingL :=$ OptLinExt($P \backslash partialL$)
    add $(c, e)$ to $TabuPositions$
    if $s(partialL) + 1 + s(remainingL) \geqslant s_\star$ then return $\varnothing$
    else return $partialL \oplus remainingL$
  else go to 4
Step 4. $remainingL := \varnothing$
  { complete the linear extension with s.-s.-greedy chains }
  while $D \neq \varnothing$
    $S :=$ strongly greedy paths in $D$
    $W :=$ semi-strongly greedy paths in $D$
    if $|S| > 0$ then
      $\pi :=$ any path from $S$
      $remainingL := remainingL \oplus C_\pi$
      if $C_\pi$ is the first chain in $remainingL$ then
        add $(c, e)$ to $TabuPositions$
    else { no strongly greedy paths }
      if $|W| = 1$ then
        $\pi :=$ the path from $W$, $remainingL := remainingL \oplus C_\pi$
        if $C_\pi$ is the first chain in $remainingL$ then
          add $(c, e)$ to $TabuPositions$
      else { several semi-strongly greedy paths }
        if $remainingL = \varnothing$ {i.e, first choice after split} then
          $\pi :=$ a random path from $W \backslash TabuPaths$;
          if not found then add $(c, e)$ to $TabuPositions$, return $\varnothing$
          { $\pi$ added to $TabuPaths$ in Alg. 4}
        else { not first chain after split } $\pi :=$ a random path from $W$
        $remainingL := remainingL \oplus C_\pi$
    $D := D(P \backslash (partialL \oplus remainingL))$
Step 5. return $partialL \oplus remainingL$
```

Algorithm 4 `TS-OptimizeJumpNumber`

Input: A poset P, the number of iterations T.

Output: $L = C_0 \oplus \ldots \oplus C_m$, a semi-strongly greedy linear extension of P.

```
Step 1. { Initialization }
  $TabuPositions := \varnothing$
  $TabuPaths := \varnothing$
  $currentL :=$ TS-CompleteLinExt$(P, \varnothing, |P|)$
  $bestL := currentL$
  $t := 0$ {current iteration}
Step 2. while $t \leqslant T$
    $bestNeighbour = \varnothing$
    { the split position for the best neighbour }
    $bestKC = 0$, $bestKE = 0$
    $t := t + 1$
Step 3.  { select the best neighbour solution }
    for $n = 1$ to $CheckedNeighbours$
      $kc :=$ a random number less than $|currentL|$
      $ke := |L[0]| + \ldots + |L[kc - 1]|$
        { such that $(kc, ke) \notin TabuPositions$ }
      $partialL :=$ first $kc$ chains of $currentL$
      $neighbourL :=$ TS-CompleteLinExt$(P, partialL,$
        $s(bestLinExt))$
      if $s(neighbourL) < s(bestNeighbour)$ then
        $bestNeighbour := neighbourL$
        $bestKC := kc$, $bestKE := |neighbourL[0..bestKC - 1]|$ { $= ke$ }
      $n := n + 1$
Step 4.  { move to the neighbour, update the result }
    if $bestNeighbour \neq \varnothing$ then
      $currentL := bestNeighbour$
      update $TabuPaths$ with $(bestKC, bestKE,$
        $currentL[bestKC - 1], currentL[bestKC])$
      if $s(currentL) < s(bestL)$ then $bestL := currentL$
Step 5. return $bestL$
```

We keep two tabu lists. $TabuPositions$ is a cyclic list containing $TabuSize$ recent split positions. More precisely, if L is split after kc chains, then the pair (kc, ke) may be added to $TabuPositions$, where $ke = |L[0]| + \ldots + |L[kc-1]|$ is the total number of poset elements in kept chains. The second tabu list, $TabuPaths$, contains the greedy paths before and after split positions. That is, whenever L is split after kc chains and completed, we add to $TabuPaths$ the quadruple $(kc, ke, L[kc-1], L[kc])$. This is motivated by fact that there are two major decision points when generating a neighbour: firstly, L is split at some position kc; secondly, one of available greedy paths in $D(P\backslash(L[0] \oplus \ldots \oplus L[kc-1]))$ is selected. $TabuPositions$ is a cyclic list, so after adding a new entry, the oldest one is removed. On the other hand, $TabuPaths$ is a static list, from which no entry is removed in the process of the algorithm.

When generating or completing a linear extension, an obvious change is made with respect to the original semi-strongly greedy algorithm: a greedy path is not allowed to be selected, if it is contained in the tabu list $TabuPaths$ associated with current split position (Step **4** of Algorithm 3). Further, while `OptLinExt` proceeds with greedy paths in systematic manner, here we always select one at random. Obviously, the implementation is also augmented to update both tabu lists in each iteration.

A split position (kc, ke) is added to $TabuPositions$ when the remaining subposet has either a strongly greedy path, or only one semi-strongly greedy path, or when its jump number is assessed as non-promising in Step **2**, or has been completed exactly in Step **3** (Algorithm 3). In other words, (kc, ke) is not tabu, if there are still some unexplored choices of greedy paths in the remaining subposet. Each choice of a greedy path is recorded in $TabuPaths$.

The most time-consuming subprocedure is the construction of an arc diagram for the remaining subposet whenever a greedy path is selected and added to L. Therefore, it is important to quickly reject those solutions whose number of jumps will not improve over the best one found in the preceding course of the algorithm. Thus, when a split point is selected and the diagram is reconstructed, we asses this choice by calculating the lower bound for the jump number of the remaining poset. It may immediately turn out that another split position should be randomized. We use a lower bound given by the following theorem.

Theorem 3.1.1 (Sysło [24]). *If $D(P)$ is an arc diagram of a poset P then $\sum_{v \in V} \max\{0, \mathrm{indeg}_P(v) - 1\} \leqslant s(P)$, where $\mathrm{indeg}_P(v)$ is the number of poset arcs coming into v.*

If the number of dummy arcs in the diagram is less than some fixed number $MaxDummies$, we apply `OptLinExt` to the remaining poset.

4. Benchmarks

In this section we benchmark the proposed tabu search algorithm on two non-trivial classes of posets.

4.1. Interval orders

In the case of interval orders, we first run `TS-OptimizeJumpNumber` and compare the quality of its solutions with optimal ones, obtained via a reduction to the subgraph packing problem. We now explain how these optimal values are computed.

The jump number of interval orders

Interval orders have a well-known characterization (see Fishburn [6]) which includes their canonical representation, that is, Algorithm 5.

Algorithm 5 Canonical representation of an interval order (see [16])

```
Input: $(P, <_P)$, an interval order.
Output: $\{I_p = [l(p), r(p)]\}_{p \in P}$, a compact family of intervals representing $(P, <_P)$.
Step 1. Sort $(\mathcal{S}_P, \supseteq)$: $Succ_1 \supseteq Succ_2 \supseteq \ldots \supseteq Succ_e = \varnothing$.
Step 2. Sort $(\mathcal{P}_P, \subseteq)$: $\varnothing = Pred_1 \subseteq Pred_2 \subseteq \ldots \subseteq Pred_e$.
Step 3. Assign to each $p \in P$ its left endpoint $l(p) = i - 1$ such that
    $Pred_i = Pred_P(p)$ and its right endpoint $r(p) = j - 1$
    such that $Succ_j = Succ_P(p)$.
```

The obtained *canonical intervals* are then written into a *table* of size $e \times e$, where $e = |\mathcal{P}_P| = |\mathcal{S}_P|$ (see Figure 4). For an interval $[l(p), r(p)]$ its corresponding element p is put in the cell in row $l(p)$, column $r(p)$. Then, successive bumps of a linear extension may be read from the table along a sequence of ordered pairs of the form $T = \{t_i = (t_{col}, t_{row})\}_{i=1,\ldots,b}$, called a *bump sequence*, where $b \leqslant e - 1$ is its length. Every bump t in T satisfies $t_{col} < t_{row}$ and for every two consecutive bumps (s, t), $s_{row} \leqslant t_{col}$.

We say that the columns and rows outside T are *omitted*. The problem is to generate a bump sequence of maximum cardinality, i.e., to decide, which rows and columns should be omitted so as to obtain a *realizable* bump sequence (so some L can be read along it). Given a realizable bump sequence of length b_T one applies a quick procedure (see [16]) to obtain a linear extension with at least b_T bumps.

Definition 4.1.1. *Graph of intervals* $G_I(P)$ takes non-empty table cells as vertices. Edges are added for vertices positioned consecutively in a column or in a row (see Example 4.1.6 and Figure 4). A component C of this graph is *unsaturated* if none of its vertices is situated on the boundary of the table (i.e., in the lowest row or in the rightmost column, or on the diagonal), nor any cell of C contains a multiple element, nor C itself contains a cycle.

The number of unsaturated components is denoted by u and is typically much smaller than $e \leqslant n$.

In [16], Mitas characterizes realizable bump sequences as follows.

Theorem 4.1.2. *For a bump sequence T to be realizable it suffices that each unsaturated component C satisfies one of two properties:*

1) *($P1$) C contains a vertex in a column or in a row which is omitted by T.*
2) *($P2$) C contains an element $[j, j+q]$ such that the columns $j, \dots, j+q-1$ and the rows $j+1, \dots, j+q$ are omitted by T.*

Theorem 4.1.2 motivates the following definition.

Definition 4.1.3. In the *graph of unsaturated components* $G_U(P)$ vertices correspond to unsaturated components of $G_I(P)$. An edge joins v_C and v_D if component C contains a vertex in column i and component D contains a vertex in row i or row $i+1$.

Then, the properties $P1$ and $P2$ of bump sequences are mapped to edges and certain odd cycles (called *valid cycles*) of $G_U(P)$. With each edge and with each valid cycle there is an associated set of pairs (col_i, row_i) or (col_i, row_{i+1}), which when removed from a bump sequence result in ($P1$) or ($P2$) being satisfied by the corresponding unsaturated components. In this way the jump number problem is reduced to a subgraph packing problem which is to find an optimal packing of vertex-disjoint edges and valid cycles. A packing involving v vertices and c cycles yields a bump sequence with $u - \frac{v+c}{2}$ lost bumps, so $v + c$ is to be maximized. We refer the reader to the article of Mitas [16] and to a recent work of the first named author [13, 14] for a detailed explanation of this reduction.

The following result is a corollary from the previous work of Mitas concerning approximation of the jump number.

Corollary 4.1.4. *The jump number of an interval order P can be computed in time $\mathcal{O}(2^n \cdot poly(n))$.*

Proof. By Mitas, for an optimal bump sequence there is an optimal packing with respect to $\frac{v+c}{2}$. It can be seen that the number of valid cycles is bounded by $e \leqslant n$. Hence, it suffices to enumerate all packings of vertex-disjoint valid cycles, and supplement each such packing H with a maximum matching M on $G_U(P) \setminus H$. From all candidate packings $H + M$ we choose the one maximizing $\frac{v+c}{2}$. □

Remark 4.1.5. Corollary 4.1.4 is an alternative proof of a more general fact, that the jump number can be computed in time dominated by 2^n for an arbitrary poset. Indeed, with any P one associates an instance of the Traveling Salesman Problem, in which vertices correspond to the points of P, and the distances (travel costs) are as follows:

- if $p <_P q$ then $c_{pq} = 0$,
- if $p \not<_P q$ then $c_{pq} = 1$,
- otherwise $c_{pq} = \infty$.

One more vertex d is added such that distances from d to the minimal elements are 0 and distances from the maximal elements to d are 0. Then, for every linear extension L there is a corresponding Hamiltonian cycle from d to d. Total cost of each such cycle is equal to $s_L(P)$ and other permutations contain a connection of cost ∞. Hence, a dynamic programming algorithm of Held and Karp [9] for TSP can be used to compute $s(P)$ in time complexity $\mathcal{O}(2^n \cdot \text{poly}(n))$.

Nonetheless, in our experiments it is beneficial to apply the algorithm described in the proof of Corollary 4.1.4. In practice, due to typical structure of arising graphs, by incorporating simple heuristics, this algorithm is of satisfactory speed. There is often a limited number of valid cycles in $G_U(P)$. Moreover, they usually overlap, so that an exhaustive search algorithm may avoid many subsets of cycles, which have at least one common vertex. Thus, the jump number can be computed in reasonable

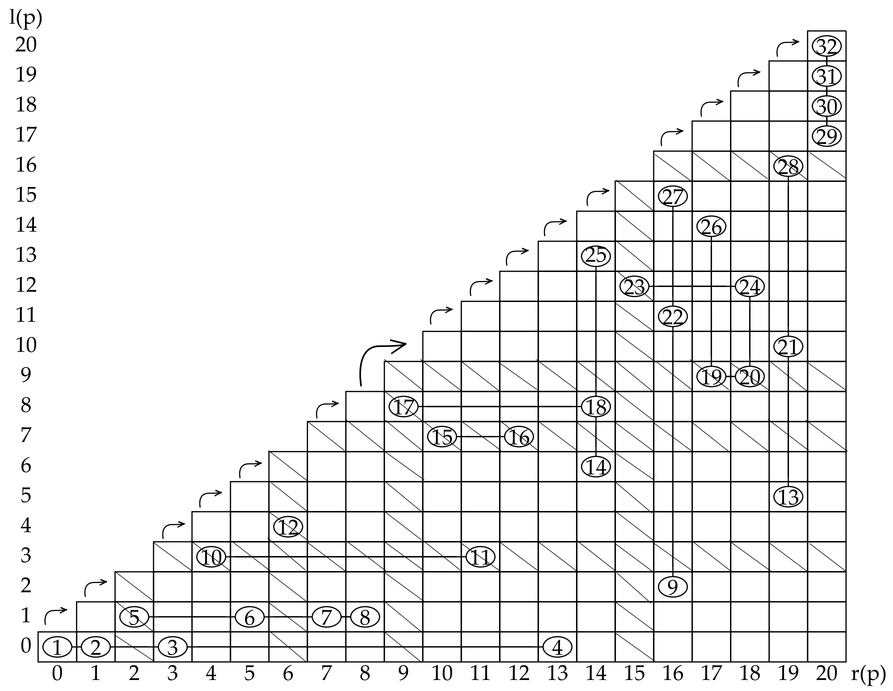

Figure 4. An interval order in canonical representation

time even for interval orders P having several hundred elements (i.e., within 10 seconds).

Example 4.1.6. An interval order in Figure 4 consists of 32 elements, with graph of intervals forming 10 components, of which 8 are unsaturated. Hence, its graph of unsaturated components (defined above) has 8 vertices, and there are three valid cycles: one pentagon, and two triangles. An optimal packing is composed of the triangle $\{\{9, 22, 27\}, \{13, 21, 28\}, \{19, 20, 23, 24, 26\}\}$ and edges $\{\{12\}, \{15, 16\}\}$, $\{\{10, 11\}, \{5, 6, 7, 8\}\}$. Corresponding bump sequence is visualised by arrows above the table, denoting successive bumps of the optimal linear extension.

Random interval orders

Our first experiment was to verify the quality of solutions generated by the proposed tabu search procedure on random interval orders. For each pair $(n, k), n \in \{100, 150, 200\}, k \in \{10, 20, \ldots, \frac{n}{2}\}$ a hundred of interval orders were randomized, consisting of n elements and containing k dummy arcs in their arc diagrams. `TS-OptimizeJumpNumber` was started for every such instance, with a limit of iterations equal to n. The algorithm was aborted upon finding an optimal solution (known a priori by Corollary 4.1.4). Every time, an optimal linear extension has been found. Amongst 100-element posets it took at worst 45 iterations (10 seconds) for an instance containing 40 dummies. Amongst 200-element posets the worst found case required 42 iterations (47 seconds) and contained 60 dummies. On average, optimum was found after 20 iterations.

Hard interval orders

In our previous research concerning approximation of the jump number on interval orders [12], a vast amount of posets were recognized, which result in suboptimality of solutions generated by formerly known algorithms. We use them to benchmark approximation algorithms of Felsner [5], Sysło [25], and Mitas [16]. Consequently, we use them now to benchmark the tabu search procedure proposed in Section 3. For example, the poset in Figure 2 contains three semi-strongly greedy paths. It is unknown which of them should be chosen in order to reach an optimal linear extension. If many similar posets are joined by series compositions, then there are very many decision points in a process of a greedy algorithm.

In our second experiment those hard interval orders were taken on 100, 150 and 200 elements. Samples of 10 results for each cardinality n are reported in Tables 1, 2, 3. Each entry corresponds to one input instance P, and contains: the number of dummy arcs in P, its jump number $s(P)$, the sequence of approximate solutions generated by `TS-OptimizeJumpNumber`, the number of iterations, the running time in seconds, and the error $\frac{s_{APX}(P)-s(P)}{s(P)}$ of the best found linear extension. If n iterations did not suffice to find the optimum, the time of last improvement in the tabu search process is also reported.

Observation 4.1.7. The proposed tabu search algorithm, applied to n-element interval orders, generates linear extensions with no more jumps than 105% of optimum, in n iterations.

4.2. Two-dimensional posets

The complexity status of the jump number problem on 2D posets is unsettled. Even though it has not been classified as NP-hard, we are attempting to solve the following Problem 4.2.1.

	d.a.	$s(P)$	$s_{APX}(P)$	iterations	time	error
P_1	28	22	$30\dots22$	32	7 s	0.0000
P_2	32	30	$39\dots30$	48	24 s	0.0000
P_3	29	24	$29\dots24$	89	25 s	0.0000
P_4	32	26	$33\dots26$	75	19 s	0.0000
P_5	28	21	$25\dots21$	18	7 s	0.0000
P_6	26	22	$26\dots22$	20	6 s	0.0000
P_7	31	28	$32\dots28$	26	10 s	0.0000
P_8	32	26	$34\dots26$	17	8 s	0.0000
P_9	35	29	$35\dots30$	100 (36)	17 s (8 s)	0.0345
P_{10}	37	31	$36\dots31$	59	38 s	0.0000

Table 1. Performance of tabu search on 100-element interval orders

	d.a.	$s(P)$	$s_{APX}(P)$	iterations	time	error
P_1	34	30	$36\dots30$	47	19 s	0.0000
P_2	46	32	$39\dots32$	14	19 s	0.0000
P_3	45	34	$42\dots35$	150 (26)	81 s (18 s)	0.0294
P_4	50	44	$55\dots46$	150 (90)	123 s (85 s)	0.0455
P_5	37	32	$39\dots32$	36	17 s	0.0000
P_6	51	42	$52\dots43$	150 (27)	35 s	0.0238
P_7	38	36	$45\dots36$	36	52 s	0.0000
P_8	42	37	$45\dots37$	52	66 s	0.0000
P_9	45	35	$42\dots35$	39	20 s	0.0000
P_{10}	43	37	$42\dots38$	150 (29)	143 s (31 s)	0.0270

Table 2. Performance of tabu search on 150-element interval orders

	d.a.	$s(P)$	$s_{APX}(P)$	iterations	time (s)	error
P_1	50	45	$52\dots46$	200 (164)	380 (336)	0.0222
P_2	54	50	$61\dots52$	200 (36)	352 (101)	0.0400
P_3	57	49	$61\dots51$	200 (13)	258 (40)	0.0408
P_4	54	49	$62\dots51$	200 (158)	415 (362)	0.0408
P_5	56	50	$62\dots52$	200 (16)	306 (43)	0.0400
P_6	56	48	$57\dots49$	200 (126)	328 (228)	0.0208
P_7	63	49	$57\dots50$	200 (66)	327 (132)	0.0204
P_8	54	48	$54\dots50$	200 (6)	160 (10)	0.0417
P_9	54	49	$59\dots50$	200 (73)	235 (109)	0.0204
P_{10}	65	53	$64\dots54$	200 (50)	413 (144)	0.0189

Table 3. Performance of tabu search on 200-element interval orders

Problem 4.2.1. Give an approximation algorithm for the jump number problem on two-dimensional posets.

For this purpose it is useful to analyze the performance of general-purpose algorithms on 2D orders. For $n > 50$ we usually fail to compute $s(P)$ exactly in reasonable time. Hence, we compare each tabu search result with a lower bound on the $s(P)$, inferred from the results of Ceroi.

The bump number of two-dimensional posets

As observed by Ceroi [4], for two-dimensional posets the jump number can be interpreted as the problem of finding a maximum weight independent set of a family of axis-parallel rectangles corresponding to certain chains of P. Let P be a two-dimensional poset with realizer $\{L_1, L_2\}$. With $p \in P$ we associate a point $(x, y) \in \mathbb{R}^2$ such that x is the position of p in L_1 and y is the position of p in L_2. Then, each chain of P is easily seen as a rectangle in $\mathbb{R}^2$. Linear extensions are formed only from chains C which are convex, i.e., $\forall p <_P q <_P r \in P$, if $p \in C$ and $r \in C$ then $q \in C$. If by $R(P) = (V_R, E_R)$ we denote the graph of rectangle intersections, in which a weight for each vertex $v \leftrightarrow C_v$ is equal to its bumps, i.e., $w(v) = |C_v| - 1$, then we have the following result.

Lemma 4.2.2 (Ceroi [4]). *The maximum bump number $b(P)$ is equal to the maximum weight of an independent set in $R(P)$.*

Hence, to calculate $b(P)$ it suffices to solve an integer linear program for maximum weight independent set (MWIS), $\max \sum_v w(v) \cdot x(v)$ s.t. $x(v) \in \{0, 1\}$ for each $v \in V_R$, and $x(u) + x(v) \leqslant 1$ for each $(u, v) \in E_R$. However, $|V_R|$ is usually greater than $|P|$ and exact computation of MWIS quickly becomes infeasible. Therefore, we only get an upper bound $b_{UB}(P) \geqslant b(P)$ by solving an LP-relaxation of this formulation, i.e., with $0 \leqslant x(v) \leqslant 1$ (and so a lower bound $s_{LB}(P) = \lceil n - 1 - b_{UB}(P) \rceil$ for the jump number).

Random two-dimensional posets

In the first experiment concerning two-dimensional posets n was set to 30. For 30-element posets optimal solution can be computed by `OptLinExt`. On larger instances we have to resort to the LP-relaxation which provides an upper bound for $b(P)$.

Tens of random 30-element two-dimensional orders were generated containing a varied number of dummy arcs (from 10 to 50). First, $s(P)$

$b(P)$	16	17	18	17	16
$b_{UB}(P)$	17.25	18.66	19.0	17.5	17.5
$b(P)$	17	19	18	16	17
$b_{UB}(P)$	18.33	19.75	19.88	17.33	18.11

Table 4. Discrepancy between the bump number and its LP relaxation, $n = 30$

	d.a.	$\|V_R\|$	$b_{UB}(P)$	$s_{LB}(P)$	$s_{APX}(P)$	error
P_1	84	341	41.5	18	$23 \ldots 19$	0.0556
P_2	81	339	39.5	20	$25 \ldots 22$	0.1000
P_3	74	375	38.5	21	$24 \ldots 22$	0.0476
P_4	66	349	41.5	18	$23 \ldots 20$	0.1111
P_5	100	337	40.75	19	$22 \ldots 19$	0.0000
P_6	90	297	39.125	20	$26 \ldots 23$	0.1500

Table 5. Performance of tabu search on 60-element 2D posets

was computed. Then, the tabu search procedure was started with a limit of n iterations. Optimum was found by `TS-OptimizeJumpNumber` in all cases beside one poset with 50 dummies, for which a linear extension was generated having 12 jumps instead of 11. On all inputs, the algorithm required at most 21 iterations and no more than 2 seconds.

By this occasion, $b(P)$ was additionally compared with the linear programming relaxation $b_{UB}(P)$ of the equivalent MWIS problem. Some typical discrepancies between these values are shown in Table 4.2. The number of vertices in $R(P)$, i.e., the number of convex chains in P, varied from 96 to 138.

In the next experiment, 150 posets were generated with $n = 60$. This time, the approximated number of jumps was compared only with the lower bound obtained via an LP relaxation of MWIS on $R(P)$, that is, $s_{LB}(P) = \lceil n - 1 - b_{UB}(P) \rceil$. The results for a sample of 6 posets are reported in Table 5.

An average error of $s_{APX}(P)$, when compared against $s_{LB}(P)$, was 0.12. In the worst spotted case, it was 0.29. The number of dummy arcs in these posets ranged from 60 to 100. The number of vertices in $R(P)$ ranged from 290 to 403. The running time was always below 12 seconds.

Finally, 30 two-dimensional posets with $n = 90$ were taken. The error of s_{APX} obtained with n iterations of our algorithm was on average 0.15 and at most 0.29. The time of optimization was always below 45 seconds

	d.a.	$\lvert V_R\rvert$	$b_{UB}(P)$	$s_{LB}(P)$	$s_{APX}(P)$	error
P_1	166	593	63.57	26	$35\ldots29$	0.1154
P_2	150	566	62.44	27	$33\ldots30$	0.1111
P_3	158	570	60.96	29	$38\ldots31$	0.0690
P_4	167	599	61.58	28	$39\ldots36$	0.2857
P_5	163	631	63.00	26	$33\ldots29$	0.1154
P_6	173	557	60.17	29	$39\ldots34$	0.1724

TABLE 6. Performance of tabu search on 90-element 2D posets

(it took much longer to compute the lower bound $s_{LB}(P)$ with linear programming). 6 representative cases are reported in Table 6.

Observation 4.2.3. The proposed tabu search algorithm, applied to n-element 2D posets, generates linear extensions with no more jumps than 130% of optimum, in n iterations.

The parameters of the tabu search in all the benchmarks were set as follows: $TabuSize = 10$, $CheckedNeighbours = 7$, $MaxDummies = 15$. These values have been established experimentally as a good compromise between the running time and convergence of the algorithm.

The running time of our tabu search algorithm could be improved by incorporating more involved methods to rebuild an arc diagram in every step. For instance, it was observed in [25] that in the case of interval orders, an arc diagram can be generated with Algorithm 5. This is much faster than Algorithm 1. We have tried this construction and it turned out that the running times reported in Section 4.1 would decrease by a factor of 3.

All the experiments have been performed on a 64-bit computer with Intel® Core™ i5-2500K CPU clocked at 3.30 GHz, and 24 GB of RAM. The algorithms have been implemented in the C# language for the .NET Framework 4. For linear programming, the `LPSolve` function from the `Optimization` package of Maple™ was employed.

4.3. An approximation ratio for two-dimensional posets

A way to measure the quality of approximation algorithms is to assess their approximation ratio.

Definition 4.3.1. An algorithm $\mathcal{A}$ is an ϵ-approximation algorithm (with $\epsilon > 1$) for a problem $\mathcal{P}$ if it runs in time polynomial in the input size and always generates a solution of value APX $\geqslant \frac{\mathrm{OPT}}{\epsilon}$ when $\mathcal{P}$ is a maximization problem, or of cost APX $\leqslant \epsilon$OPT when $\mathcal{P}$ is a minimization problem.

We now look again at the jump number problem on 2D orders via the Ceroi reduction, described in Lemma 4.2.2 of Section 4.2. The bump maximization problem can be solved by computing a maximum weight independent set of rectangles in $R(P) = (V_R, E_R)$. Some approximation algorithms are known for this problem.

Theorem 4.3.2 (Agarwal, van Kreveld, Suri [1]). *There exists a* $\log|V|$*-approximation algorithm for the maximum weight independent set problem on intersection graphs of rectangles.*

The original paper of Agarwal et al. focuses on maximum independent set problem, but their algorithm extends to the weighted variant in a straightforward way, see [11, pp. 136–140]. All logarithms are in base 2.

So this implies an approximation of the bump number. Can this result be used to approximate the jump number, too? We claim that the answer is positive.

Theorem 4.3.3. *There exists an* $(n/\log\log n)$*-approximation algorithm for the jump number of two-dimensional posets.*

In the proof, we use the following result.

Theorem 4.3.4 (McCartin [15]). *There exists an algorithm to decide for any poset* P *whether* $s(P) \leqslant h$*, running in time* $\mathcal{O}(h^2 h! n)$*.*

Proof of Theorem 4.3.3. The algorithm computes an approximate bump number $b_A(P) \geqslant \frac{b(P)}{\log|V|}$, where $|V|$ is the number of vertices in $G(R)$. We obtain an approximate jump number $s_A(P) = n - 1 - b_A(P)$, so the approximation ratio is

$$\frac{s_A(P)}{s(P)} \leqslant \frac{n - 1 - \frac{b(P)}{\log|V|}}{n - 1 - b(P)}.$$

We verify when this ratio is worse than $\frac{n}{\log\log n}$:

$$\frac{n - 1 - \frac{b(P)}{\log|V|}}{n - 1 - b(P)} > \frac{n}{\log\log n},$$

$$\log\log n \cdot (n - 1 - \frac{b(P)}{\log|V|}) > n \cdot s(P),$$

$$n \cdot s(P) < (n-1) \cdot \log\log n - \frac{\log\log n \cdot b(P)}{\log |V|} < n \cdot \log\log n,$$

$$s(P) < \log\log n.$$

When the jump number is very small, i.e., $s(P) < \log\log n$, the algorithm of Agarwal et al. [1] does not provide the claimed approximation of $s(P)$, but in such cases we may compute the jump number to optimality by the McCartin algorithm [15] in polynomial time, since one quickly verifies that $(\log\log n)! < n$. □

5. Conclusions and future work

In this paper a new tabu search algorithm for the jump number problem has been proposed and benchmarked. There are some remarks concerning the approximation, exact computation and hardness of the problem.

The results of our algorithm on interval orders have been compared with optimal values. The experiments reveal that in relatively short time linear extensions can be obtained with no more jumps than 105% of $s(P)$. It is easy to spot interval orders for which a 50% suboptimal semi-strongly greedy linear extension exists [12]. Hence, it is definitely worth to apply tabu search to improve the quality of generated solutions.

Previously, no approximation algorithms have been given for the class of two-dimensional posets. Our work is an attempt to fulfill this demand. In comparison with a lower bound on the jump number, linear extensions generated by our tabu search procedure turn out to be at most 30% suboptimal. Since the LP relaxation is usually inexact, the real error is probably lower. We have proved by other techniques that there is an $(n/\log\log n)$-approximation algorithm for the jump number problem on two-dimensional posets We hypothesize that even stronger approximation ratio could be proved for this class, perhaps by a detailed analysis of the semi-strongly greedy algorithm. Addressing this questions is our main objective in the future work. More precisely, it is conceivable that such an algorithm may be based on the rectangle intersection MWIS lower bound.

We have also empirically verified the number of all linear extensions generated by the exact algorithm `OptLinExt` of Sysło. It turns out that their number is far from the theoretical bound of $k!$. Considering that we have proved, based on the results of Mitas, that the complexity of computing $s(P)$ on interval orders is dominated by 2^n, and is lower in practice, we believe that an estimation of the running time of `OptLinExt` can be improved in this class. Another question for further research is, whether or not the time complexity bound of this algorithm can be lowered on arbitrary posets.

The complexity status of computing $s(P)$ on two dimensional orders is open. Our experiments show that the space of semi-strongly greedy linear extensions is explicitly greater than in the case of interval orders. Hence, from this point of view, the class of two dimensional posets seems harder than interval orders for the jump number problem.

References

[1] P.K. Agarwal, M. van Kreveld, S. Suri, *Label placement by maximum independent set in rectangles*, Comput. Geom. 11 (1998), 209–218.

[2] A. Arnim, C. Higuera, *Computing the jump number on semi-orders is polynomial*, Discrete Appl. Math. **51**, 219–232 (1994).

[3] V. Bouchitté, M. Habib, *NP-completeness properties about linear extensions*, Order **4** (1987), 143–154.

[4] S. Ceroi, *A weighted version of the jump number problem on two-dimensional orders is NP-complete*, Order **20** (2003), 1–11.

[5] S. Felsner, *A 3/2-approximation algorithm for the jump number of interval orders*, Order **6** (1990), 325–334.

[6] P.C. Fishburn, *Interval orders and interval graphs. A study of partially ordered sets*, Wiley, New York, 1985.

[7] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.

[8] M. Habib, *Comparability invariants*, in: M. Pouzet, D. Richard, ed., *Orders: Description and Roles*, Annals of Discrete Mathematics 23 (1984), pp. 371–386.

[9] M. Held, R.M. Karp, *A dynamic programming approach to sequencing problems*, J. Soc. Ind. Appl. Math. **10** (1962), 196–210.

[10] C. Higuera, L. Nourine, *Drawing and encoding two-dimensional posets*, Theor. Comput. Sci. **175** (1997), 293–308.

[11] C. Iturriaga, *Map labeling problems*, PhD Thesis, University of Waterloo, 1999.

[12] P. Krysztowiak, M.M. Sysło, *An experimental study of approximation algorithms for the jump number problem on interval orders*, Discrete Appl. Math., submitted (2012).

[13] P. Krysztowiak, *An improved approximation ratio for the jump number problem on interval orders*, Theor. Comput. Sci **513** (2013), 77–84.

[14] P. Krysztowiak, *Improved approximation algorithm for the jump number of interval orders*, Electron. Notes Discrete Math. **40** (2013), 193–198.

[15] C. McCartin, *An improved algorithm for the jump number problem*, Inform. Process. Lett. **79** (2001), 87–92.

[16] J. Mitas, *Tackling the jump number of interval orders*, Order **8** (1991), 115–132.

[17] A. Ngom, *Genetic algorithm for the jump number scheduling problem*, Order **15** (1998), 59–73.

[18] W.R. Pulleyblank, *On minimizing setups in precedence-constrained scheduling*, Report No. 81185 – OR (unpublished), May 1981.

[19] I. Rival, *Optimal linear extensions by interchanging chains*, Proc. Am. Math. Soc. **89** (1983), 387–394.

[20] J. Spinrad, J. Valdes, *Recognition and isomorphism of two dimensional partial orders*, Automata, Languages and Programming, LNCS 154 (1983), 676–686.

[21] M.M. Sysło, *A graph-theoretic approach to the jump-number problem*, in: I. Rival, ed., *Graphs and Orders*, D. Reidel, Dodrecht 1985, pp. 185–215.

[22] M.M. Sysło, *Minimizing the jump number for partially ordered sets: a graph-theoretic approach*, Order **1** (1984), 7–19.

[23] M.M. Sysło, *Minimizing the jump number for partially-ordered sets: a graph-theoretic approach, II*, Discrete Math. **63** (1987), 279–295.

[24] M.M. Sysło, *An algorithm for solving the jump number problem*, Discrete Math. **72** (1988), 337–346.

[25] M.M. Sysło, *The jump number problem on interval orders: A 3/2 approximation algorithm*, Discrete Math. **144** (1995), 119–130.

[26] M.H. El-Zahar, J.H. Schmerl, *On the size of jump-critical ordered sets*, Order **1** (1984), 3–5.

[27] M.H. El-Zahar, I. Rival, *Examples of jump-critical ordered sets*, SIAM J. Algebr. Discrete Meth. **6** (1985), 713–720.

[28] M.H. El-Zahar, *On jump-critical posets with jump-number equal to width*, Order **17** (2000), 93–101.

Contact information

P. Krysztowiak,
M. M. Sysło

Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University, Toruń, Poland
E-Mail(s): pk@mat.umk.pl,
syslo@mat.umk.pl

Received by the editors: 29.11.2013
and in final form 29.11.2013.