Annales UMCS Informatica AI XI, 1 (2011) 15–22
DOI: 10.2478/v10065-011-0025-4

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

The database of interval orders difficult for the jump number minimizing algorithms

Przemysław Krysztowiak[1]*

[1] *Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Chopina 12/18, 87-100 Toruń, Poland*

Abstract – The problems of scheduling jobs on a single machine subject to precedence constraints can often be modelled as the jump number problem for posets, where a linear extension of a given partial order is to be found which minimizes the number of noncomparabilities. In this paper, we are investigating a restricted class of posets, called interval orders, admitting approximation algorithms for the jump number problem, in which the problem remains NP-complete. We have implemented three known approximation algorithms for this problem, all of which are guaranteed to produce solutions that are at most 50% worse than the optimal ones. More importantly, we have performed an exhaustive search for particularly hard interval orders, which enforce the algorithms to generate orderings which are exactly 50% worse than the optimal linear extensions. The main purpose of this paper is to present the database of those problematic posets.

1 Introduction

The jump number problem for posets consists in determining a linear extension with a minimum number of adjacent pairs that are incomparable in a given poset. Possible applications of this NP-hard optimization problem can be found in the area of task scheduling with precedence constraints. Three polynomial-time $\frac{3}{2}$-approximation algorithms are known in a restricted class of interval orders, where the problem remains NP-complete. These algorithms (published by Felsner [**1**], Sysło [**2**] and Mitas [**3**]) have been implemented and intensively tested. A database of hard instances has been obtained on which the worst deviation from optimum is achieved in pessimistic runs of each of the known algorithms. Our results are invaluable for researchers interested in developing new algorithmic ideas, since a great amount of nontrivial sample posets can

*pk@mat.umk.pl

both help the attracted individuals come up with new concepts and restrain them from following unfortunate thoughts. Moreover, our findings give themselves an interesting insight into the problem.

The paper begins with a formal problem statement and an example application in task scheduling. Then, the necessary background material on interval orders follows. After that, we introduce our methodology of tackling the problem, and review the approximation algorithms under investigation. Finally, our genetic procedure of obtaining particularly difficult interval orders is presented, together with an elaboration concerning the database for storing those posets.

2 Problem statement

By $(P, \leqslant_P)$, or simply by P, we denote a finite *poset* of *size* $|P| = n$ (recall that a poset, or a partially ordered set, consists of a set P together with a binary relation $\leqslant_P$ which is reflexive, antisymmetric, and transitive) . We write $p <_P q$ if $p \leqslant_P q$ and $p \neq q$. For any $p \in P$, $Succ(p) = \{q \in P : p <_P q\}$ denotes the set of *successors* of p and $Pred(p) = \{q \in P : q <_P p\}$ denotes the set of *predecessors* of p. A *family of distinct successors sets* in a poset $(P, \leqslant_P)$ is denoted by $\mathbb{S} = \{Succ(p) : p \in P\}$ and a *family of distinct predecessors sets* is denoted by $\mathbb{P} = \{Pred(p) : p \in P\}$. $Min(P)$ contains the *minimal elements* in $(P, \leqslant_P)$, i.e., the elements with no predecessors.

By a *linear extension* of a given poset we mean a total ordering of its elements, $L = x_1, x_2, \ldots, x_n$, such that $x_i \leqslant_P x_j$ implies $i < j$ for all i, j. Two adjacent elements x_i, x_{i+1} in a given linear extension L of a poset P are separated with a *jump* if $x \not\leqslant_P x_{i+1}$; otherwise, they are separated with a *bump*. The *jump number problem* consists of determining a linear extension L with the minimum number of jumps, denoted by $s(P) = \min\{s(L, P) | L \text{ is a linear extension of } P\}$. Obviously, the problem is equivalent to maximization of the number of bumps, i.e., to determining $b(P) = \max\{b(L, P) | L \text{ is a linear extension of } P\}$, since for any linear extension L we have $s(L, P) + b(L, P) = n - 1$. The problem of minimizing $s(P)$ is NP-complete even in the class of interval orders [**3**]. Notably, when only unit length intervals are allowed in a poset representation, the problem admits polynomial-time solution [**4**].

3 Example application

One possible occurrence of the jump number problem can be found in the area of task scheduling. Suppose a set of computer programs (or tasks) is to be sequenced for an execution on a one-processor machine. Assume that some of these programs require on input the results of other programs. It is easy to see that these precedence constraints define a partial order on the set of involved computer programs to be scheduled. Additionally, suppose that two of the tasks, expected to be run one after another, are sequenced in different order. This means that the required results of the first task need to be stored (e.g., in a database) so that they are available later when

they are needed for the second task. Since every database access has a fixed cost, we are interested in a schedule for all the tasks that minimizes the number of incomparabilities against the resulting partial order. In this way a jump-optimal linear extension can be seen as the most efficient sequence of executing all the tasks.

4 Interval orders

A poset $(P, \leqslant_P)$ is an *interval order* if a bijective mapping exists between its elements and closed intervals on the real line, $P \leftrightarrow \{I_x = [l(x), r(x)]\}_{x \in P}$, satisfying $x \leqslant_P y$ if and only if $r(x) < l(y)$ ($l(x)$ and $r(x)$ are understood respectively as left and right endpoint of an interval corresponding to $x \in P$). It is well known that interval orders can be characterized as posets without two independent 2-element chains as induced subposet. Moreover, in the case of interval posets, $(\mathbb{P}, \subset)$ and $(\mathbb{S}, \subset)$ are linear orders. The proofs of these results, as well as the additional background material on interval orders, can be found in the monograph [**5**] (see also [**3**]).

The above facts justify a *canonical representation* of a given interval order P: sort the distinct successors sets by inclusion, $Succ_1 \supseteq Succ_2 \supseteq \ldots \supseteq Succ_m = \emptyset$, and sort the distinct predecessors sets by inclusion $\emptyset \subseteq Pred_1 \subseteq Pred_2 \subseteq \ldots \subseteq Pred_m$. Assign to every $x \in P$ the left endpoint $l(x) = i - 1$ such that $Pred_i = Pred(x)$ and the right endpoint $r(x) = j - 1$ such that $Succ_j = Succ(x)$. In this way a family of canonical intervals is obtained, which are further used in the Sysło and Mitas algorithms. In the Mitas algorithm canonical intervals are written into a square *table*: an interval $[l(x), r(x)]$ is put into a cell in row $l(x)$ and column $r(x)$. Note that when more poset elements are assigned, the same interval, respective cell contains them all. The size m of the table is called a *magnitude* of an interval order and is equal to the number of different successors sets, i.e., $m = |\mathbb{P}| = |\mathbb{S}|$. On the other hand, in the algorithm of Sysło there is produced an *arc diagram* whose arcs are initialized with canonical intervals. We illustrate these two representations in Fig. 1, where an examplary poset is additionally pictured as the well-known Hasse diagram. The algorithms of Felsner, Sysło and Mitas are reviewed in more details in the subsequent sections.

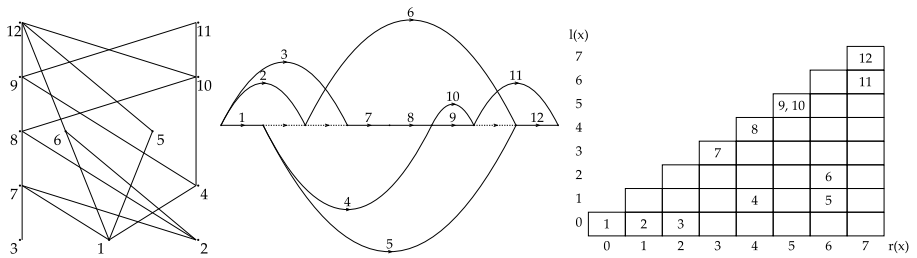

Fig. 1. An examplary interval poset represented in the three ways: a Hasse diagram, an arc diagram, and a table.

5 Methodology

We are now ready to sketch the three known algorithms which are the subject of our considerations and were used to obtain the difficult instances. After a short introduction of tested procedures we describe our methodology of obtaining those interval orders where the approximation ratio is stretched to maximum.

5.1 The algorithm of Felsner

The algorithm proposed by Felsner [**1**] is a direct extension of the following *generic greedy algorithm*: whenever possible, choose a minimal predecessor of a previously selected element.

We define the set $SMin(P) = \{x \in P : Succ(y) \subseteq Succ(x) \text{ for all } y \in P\}$ of poset elements with the greatest set of successors. As the reader will note, $SMin(P) \subseteq Min(P)$. The algorithm is based on the observation that an optimal linear extension always exists which begins with an element from this set. The Felsner algorithm builds a linear extension greedily, applying the following rules as long as the input poset is nonempty:

- If there are no successors of the previously selected element which are minimal, then choose an element from $SMin(P)$, add it to the generated linear extension L and remove it from P.
- If there are minimal successors of the previously added element, but they are not in $SMin(P)$, then choose one of them, add it to L and remove it from P.
- If there are successors of the previously added element in $SMin(P)$, then choose one of them, add it to L and remove it from P.

5.2 The algorithm of Sysło

Similarly to the Felsner algorithm, the Sysło algorithm ([**2, 6**]) realizes a greedy strategy. However, instead of a set representation of an interval order, directed graphs are used. There are two types of arcs in such a digraph. The poset arcs correspond to the poset elements whereas the remaining ones (presented with dashed lines in the pictures) are added to keep the relations along paths in a digraph. An *arc diagram* for an input interval order P is based on the canonical intervals representing P. In order to obtain this representation for a given interval poset, we proceed as described in the canonicalization procedure, but a copy of the whole poset is added to $\mathbb{S}$. As a result, right endpoints of intervals are increased by one and there are no one-point (degenerated) intervals. The set of vertices is taken as $V = \{0, 1, \ldots, m\}$, where $m = |\mathbb{S}| = |\mathbb{P}|$, and for every poset element x we add an arc $a_x = (l(x), r(x))$ to the set A of the arcs. Additionally, dummy arcs are added, $(i, i+1)$ for $i = 1, 2, \ldots, m-2$, whenever $(i, i+1)$ is not a poset arc. Let us denote by $h, t : A \to V$ the incidence mappings of an arc diagram, pointing to the *head* and to the *tail* of an arc respectively.

The algorithm successively finds special paths in the arc diagram and removes respective elements, augumenting the generated linear extension. There are two important

types of greedy paths, introduced by Sysło and accompanied with the proof that there is always an optimal linear extension which starts with one of such paths. In this way a generic greedy algorithm is improved because the search space becomes restricted.

A *greedy path* $\pi = (x_1, x_2, \ldots, x_f)$ in a canonical arc diagram for an interval order is defined by the following predicates:

- no $h(x_i)$, $i = 1, 2, \ldots, f-1$, is a head of an arc different from x_i and either P does not contain other elements than those in π or there is an arc y (possibly a dummy one) such that $y \neq x_f$ and $h(y) = h(x_f)$,
- each of the arcs $x_i, i = 1, 2, \ldots, f$, is a poset arc,
- canonical intervals of P corresponding to all arcs of the path are of length 1 (all but possibly the last one of them).

A *strongly greedy path* additionally either terminates in the sink of the diagram or $h(\pi)$ is a head of a poset arc b, $b \neq x_f$, such that no vertex of $\{0, 1, \ldots, t(b)\}$ is incident with a dummy arc. If a greedy path is not strongly greedy and it crosses a vertex which is a tail of a dummy arc, but not a head of a dummy arc, then we call it a *semi-strongly greedy path*.

We are now ready to present the algorithm of Sysło: Determine all greedy paths in the arc diagram for P. If a strongly-greedy path exists, take consecutive elements from this path to L, and remove this path from the diagram. Otherwise, choose any of semi-strongly greedy paths and proceed similarly.

5.3 The algorithm of Mitas

During the execution of the Mitas algorithm a table representation of a given interval order $(P, \leqslant_P)$ is exploited. The columns of this table are numbered from 0 to $m-1$ and the same concerns the rows, where $m = |\mathbb{S}| = |\mathbb{P}|$ is a magnitude of P. To get an idea of reading a linear extension from the table, it is useful to follow the basic algorithm from the Mitas paper with an examplary poset. A linear extension can be generated according to the following natural rules: add to L all elements from column 0, followed by any element from row 1. Then, take all the remaining elements from column 1, followed by any element from row 2. Continue this procedure until the whole table becomes empty.

Observe that this simple procedure does not necessarily maximize bumps in the generated linear extension. A more involved variant of the algorithm reduces the task to the cardinality matching problem. We now bring in the crucial steps of this reduction. First, a *graph of cells* is defined, in which the vertices correspond to nonempty table cells, and the edges join neighbouring nonempty cells lying in the same row or column. We call a component C of this graph *unsaturated* if none of its vertices is positioned on the boundary of the table, nor any cell of C contains a multiple poset element, nor C itself contains a cycle. It has been shown by Mitas that the unsaturated components constitute the core of the problem. Next, following Mitas, a *graph of components* is constructed, where unsaturated components C and D are connected with an edge if C

covers the i-th column and D covers the i-th row or $(i+1)$-th row of the table. A maximum matching determined on this graph is interpreted as follows: two vertices (i.e., unsaturated components) which are connected by a matched edge can be maintained with one lost bump, whereas unmatched vertices (components) have to be maintained with one lost bump for each of them. The detailed procedure of obtaining a linear extension from the table can be found in the original article of Mitas [**3**].

5.4 Searching for hard instances

We are now in the position to elucidate how the main aim of this work has been achieved. Our objective was to determine a collection of difficult interval orders, enforcing the algorithms of Felsner, Sysło and Mitas to produce orderings which are exactly 50% worse than optimal linear extensions. We have decided to adapt the genetic algorithm for this purpose. Whenever a hard poset of size n needs to be found, a chromosome consists of a collection of n intervals. The crossover operator mixes two chromosomes and produces two new chromosomes. This procedure is realized as follows: randomize an integer r in range 1 to n; the first r intervals of the first parent $R1$ are copied to a newly created child $C1$, and the rest of intervals from $R1$ are copied to a newly created child $C2$. Then, the intervals $(r+1)$ to n in $R2$ are copied to $C1$ and analogously the missing intervals, from $(r+1)$ to n in $R1$, are copied to $C2$. In the mutation procedure a randomly chosen interval is removed and another one is produced. We have tried to generate the random intervals from both the uniform distribution and the exponential distributions. It turned out that the choice of probability distribution is irrelevant for our purposes.

In a discussion concerning the time required to obtain the most difficult interval orders it is necessary to realize how the optimized objective function works. It is natural to define this function (also called the *fitness function*) in a way that it compares a solution obtained with an approximation algorithm to an optimal jump number and calculates an approximation ratio on actual interval order represented by the chromosome. Obviously, it needs to be executed after each application of mutation or crossover. Thankfully, a full inspection of solutions restricted to all semi-strongly greedy linear extensions can be performed realtively quickly. Note that there are, in fact, three similar fitness functions, each of which benchmarks another algorithm. In practice, we were able to obtain a 50%-deviated poset of the fixed size $n \leqslant 30$ in about half an hour on a 3GHz home PC when either the algorithm of Sysło or the algorithm of Felsner was inspected. The Mitas algorithm requires more time when $n > 20$ due to its computational complexity (computing the maximum matching is the most expensive part of this procedure). All algorithms considered in this paper have been implemented in the C# programming language.

6 The database of interval orders

The difficult interval orders found by our algorithm are stored in a dedicated database, which has been developed according to the following considerations. A single table is enough to accommodate all the posets. An interval order can be represented by a collection of canonical intervals defininig the posets, which are stored as one string of characters, starting with one number n (the poset size), followed by the consecutive intervals (each of which consists of two natural numbers). Therefore, the field `INTERVALS[NVARCHAR(MAX)]` contains the concatenation of these quantities and its content is sufficient to restore a poset. It is also useful to keep the poset size in a separate field `SIZE[INT]` so as to serve requests for posets of a given size with ease. Besides, an optimal linear extension is stored along with its jump number, as well as the best solution known so far (which can be generated for instance with the Mitas algorithm). We therefore have the following four fields: `OPT_JUMP_NUMBER[INT]`, `OPT_LIN_EXT[NVARCHAR(MAX)]`, `BEST_JUMP_NUMBER[INT]`, `BEST_LIN_EXT[NVARCHAR(MAX)]`. In addition, it is useful to remember which of the algorithms results in the 50%-deviated linear extension.

We also find it is useful to have twofold interface to the database, which is both visual and programmatic. On the visual side, available through a dedicated web page, all the hard interval orders stored in the database can be seen and downloaded. A user can look at the intervals constituting a poset, together with two images: a canonical arc diagram, and a table. On the programmatic side, a poset can be requested of supplied size, which is difficult for the algorithms known so far. The requested size can exceed the size of the biggest poset stored in the database. In such a case, existing interval orders are composed (by serial composition) and a big instance is returned to the user. Serial composition of canonical interval order can be realized by placing the intervals of Q just after the rightmost interval of P.

The database of difficult posets has been published on the author's website [**7**] so that everyone interested in developing new algorithms has access to the worst posets, troublesome for the present algorithms. A tiny excerpt of the posets stored in this database can be seen in Fig. 2. The first of the interval orders admits a linear extension with 2 jumps whereas the Felsner algorithm generates a solution with 3 jumps. For the second of the shown posets a linear extension with 4 jumps is possible, but the Sysło algorithm finds one with 6 jumps. The last instance results in 3 jumps when the Mitas algorithm is run, whereas a solution with 2 jumps can be found.

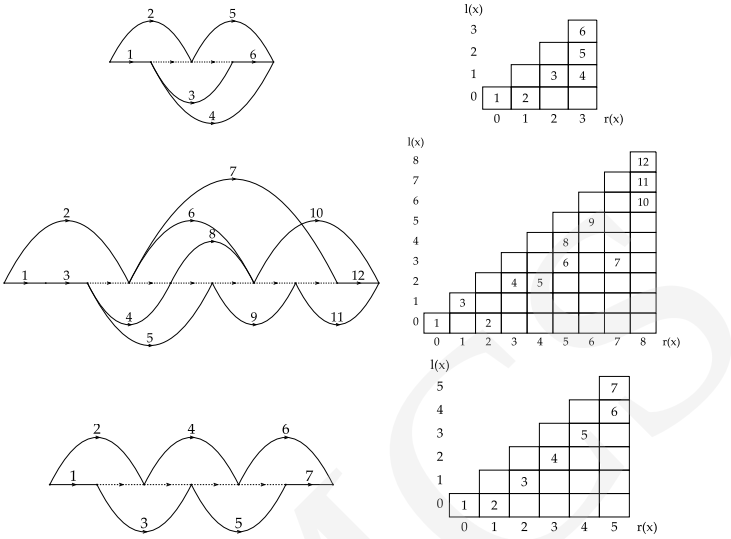

Fig. 2. Examplary difficult posets for the algorithms of Felsner, Sysło, and Mitas, respectively.

7 Future work

A generator of interval orders which behave particularly bad has been introduced and a vast collection of difficult posets has been obtained that are useful for researchers interested in developing new algorithms for the jump number problem. In addition, our knowledge of existing algorithms has been broadened. In the future the database might be re-designed to accommodate arbitrary posets and not just interval orders. Moreover, our database could be enhanced with a possibility to assimilate the posets supplied by the users.

References

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

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

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

[4] Arnim A., Higuera C., *Computing the jump number on semi-orders is polynomial*, Discrete Applied Mathematics 51 (1994): 219.

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

[6] Sysło M.M., *An algorithm for solving the jump number problem*, Discrete Mathematics 72 (1988): 337.

[7] Krysztowiak P., *http://www.mat.umk.pl/~nb/*