Available online at www.sciencedirect.com
SCIENCE DIRECT®
Discrete Applied Mathematics 154 (2006) 2097–2116
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments

Irène Charon, Olivier Hudry*

École nationale supérieure des télécommunications, 46, rue Barrault, 75634 Paris cedex 13, France

Received 23 October 2002; received in revised form 20 October 2003; accepted 21 April 2005
Available online 24 July 2006

Abstract

The linear ordering problem consists in finding a linear order at minimum remoteness from a weighted tournament T, the remoteness being the sum of the weights of the arcs that we must reverse in T to transform it into a linear order. This problem, also known as the search of a median order, or of a maximum acyclic subdigraph, or of a maximum consistent set, or of a minimum feedback arc set, is NP-hard; when all the weights of T are equal to 1, the linear ordering problem is the same as Slater's problem. In this paper, we describe the principles and the results of an exact method designed to solve the linear ordering problem for any weighted tournament. This method, of which the corresponding software is freely available at the URL address http://www.enst.fr/~charon/tournament/median.html, is based upon a branch-and-bound search with a Lagrangean relaxation as the evaluation function and a noising method for computing the initial bound. Other components are designed to reduce the BB-search-tree.

Keywords: Linear ordering problem; Slater's problem; Kemeny's problem; Median order; Tournaments; Branch and bound; Lagrangean relaxation; Noising methods

1. Introduction

A *tournament* $T = (X, E)$ is a directed graph such that for any vertex x and vertex y with $x \leqslant y$, there is one and only one arc (x, y) or (y, x) in E (for the basic definitions and results upon tournaments, see [51]). A non-negatively weighted tournament is a tournament for which a weight function w is defined from E to the set of non-negative integers (notice that, for our problem, a negative weight on an arc (x, y) would be the same as replacing (x, y) by (y, x) with the opposite weight). In the following, $T = (X, E)$ will denote a tournament weighted by such a function w and $n = |X|$ will denote the number of vertices of T. Thus, we set: $X = \{x_1, x_2, \ldots, x_n\}$. The weight of the arc (x_i, x_j) will be noted $w_{i,j}$. A linear order O defined on X will be noted $O = x_{i_1} > x_{i_2} > \cdots x_{i_n}$; as a linear order O can be represented by a directed graph (such a digraph is in fact a transitive tournament and conversely), the notation $x_i > x_j$ or similarly $x_i O x_j$ can be interpreted as orienting the arc between x_i and x_j from x_i to x_j. For any linear order $O = x_{i_1} > x_{i_2} > \cdots > x_{i_n}$

* Corresponding author. Tel.: +33 1 45 81 77 77/63: fax: +33 1 45 81 31 19.
E-mail address: hudry@inf.enst.fr (O. Hudry).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.04.020

defined on X, we define the remoteness (see [7]) $f(O)$ between O and T by

$$f(O) = \sum_{\substack{(x_j,x_i)\in E \\ x_i O x_j}} w_{j,i}.$$

This quantity $f(O)$ can be seen as the sum of the weights of the arcs that do not have the same orientation in O as in T, or similarly the sum of the weights of the arcs that we must reverse in T to get the linear order O.

The problem that we deal with in this paper, called *the linear ordering problem*, or *the maximum acyclic subdigraph problem*, *the maximum consistent arc set*, or also *the median order problem*, consists in finding a linear order (called a *median order*, see [7]) minimizing f. This problem arises in different fields, namely the social sciences, electrical engineering, agronomy, mathematics, ... (see for example [3,4,6–8,23,36,37,40,55] for references); some examples are given below:

- Thus, when all the weights belong to {0, 1}, we get (see [18] for instance) the *Feedback Arc Set problem* stated as follows: given a directed graph G, find a subset of arcs with a minimum cardinality containing at least one arc from every circuit in G. This problem is NP-hard [42].
- Or, when all the weights are equal to 1, we get the problem of Slater [57] of fitting a tournament into a linear order: given a tournament T, find a subset of arcs of T with a minimum cardinality such that reversing these arcs in T turns T into a transitive tournament (that is, a linear order called a *Slater order* of T). Some recent results, obtained when this article was under the submission process for being published, allow to show that Slater's problem is also NP-hard.
- The last example that we give here comes from voting theory and corresponds to the problem of Kemeny [43]: let X be a set of candidates on which m voters define m linear orders as their individual preferences; it is usual (see [7]) to aggregate these m linear orders into a weighted tournament T: a vertex represents a candidate and there is an arc from x_i to x_j when a majority of voters prefer candidate x_i to candidate x_j; the arc (x_i, x_j) may be weighted by the difference $w_{i,j}$ between the number of voters preferring x_i to x_j and the number of voters preferring x_j to x_i. In this case, finding the linear order (the collective preference) minimizing the number of disagreements with respect to the individual preferences is the problem above where $f(O)$ measures the number of disagreements between the m individual preferences (summarized by T) and the collective preference represented by O. It would have been possible to maximize the number of agreements, or to maximize the difference between the number of agreements and the number of disagreements: all these problems are equivalent if we assume that the individual preferences are linear orders. Kemeny's problem is NP-hard (see for instance [38]).

In what follows, we will pay attention also to the case for which all the weights are equal to 1 (Slater's problem); in this case the minimum value of f is called the *Slater index* of T and is noted $i(T)$; thus, $i(T)$ is the minimum number of arcs that we must reverse in T to get a linear order.

The aim of our work is to design a method which gives an exact solution (a median order) for tournaments of size as big as possible. For bigger tournaments, it gives an approximate solution (and so an upper bound of the optimum value) found by a noising method, and a lower bound of the optimum value computed by the Lagrangean relaxation. We describe here the principles and the performances of this method.

As minimizing f over the set of linear orders is NP-hard, we use a branch-and-bound method to obtain the optimum value (Section 4); note that this method can be adapted easily in order to find all the optimum solutions, if necessary (as in [5]). The most important way to cut the nodes of the search-tree is based on a Lagrangean relaxation (Sections 2 and 4.5). This evaluation needs much CPU time; so, in order to save time, we apply several strategies which often allow us to cut a node of the BB-search-tree before running a relaxation. To get an approximate solution, we use a noising method (Section 3). For problems small enough to be solved exactly, the noising method almost always finds the best solution in a short time. So, we hope that for bigger problems, the noising method finds solutions quite near the optimum. The Lagrangean relaxation and the noising method depend on several parameters that must be tuned before these methods may run. We tried to fix all these parameters as functions of few features of the tournament to be solved: its number of vertices, its weights and its index of transitivity suggested by Kendall and Babington Smith ([44]; see Section 5). So, the user has nothing to do in order to run the method, except to give the number of vertices, the orientations and the weights of the arcs of his/her tournament. Nevertheless, instead of the values given in Section 5, the user may give other values to the parameters in order to try to improve the method. In particular, the noising method

is better as the number of iterations grows: giving more time (that is, giving more iterations) increases the probability of getting an optimal solution. The performances of the different components of the method are given at the end of the paper (Section 6), just before the global conclusions (Section 7).

Software designed from this branch-and-bound method is freely accessible at the website of Irène Charon, at the following URL address: http://www.enst.fr/~charon/tournament/median.html.

2. Application of the Lagrangean relaxation to the linear ordering problem

This part of our work is based on a paper written by Arditti [2] who studied the application of the Lagrangean relaxation to the linear ordering problem. He did not include it into a branch-and-bound method, but used it just as a heuristic. His approach is improved and generalized here in order to get an evaluation of the nodes of the search-tree in the branch-and-bound process. In this section, we explain how the relaxation can be applied to the root of the search-tree; in Section 4, we will see how it is used inside the branch-and-bound process.

To apply the relaxation, the first thing to do is to determine which constraints will be relaxed (for a general presentation of Lagrangean relaxation, see for example [9,50]). Here, given a weighted tournament $T = (X, E)$, we look for a linear order, that is a transitive tournament, defined on X which minimizes the objective function f defined in Section 1. The relaxed constraints are the ones describing transitivity: the problem will be transformed into the research of a directed, complete, loopless and anti-symmetric graph (i.e., a tournament) which minimizes the Lagrangean function. It is now necessary to write the problem by means of equations.

If $H = (X, F)$ is a tournament defined on X, we set, for any $(x_i, x_j) \in X^2$,

$$\begin{cases} r_{i,j}(H) = 1 & \text{if } (x_i, x_j) \in F, \\ r_{i,j}(H) = 0 & \text{otherwise.} \end{cases}$$

Then, for any tournament H, we have: $\forall i \neq j$, $r_{i,j}(H) + r_{j,i}(H) = 1$ and $\forall i$, $r_{i,i}(H) = 0$. Moreover, we set $w_{i,j} = 0$ if $(x_i, x_j) \notin E$. We may generalize the definition of f (defined in Section 1 only for linear orders) to any tournament H by

$$f(H) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{j,i} \cdot r_{i,j}(H) = \sum_{i=2}^{n} \sum_{j=1}^{i-1} w_{j,i} \cdot r_{i,j}(H) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{j,i} \cdot r_{i,j}(H).$$

As we have the following equalities:

$$\begin{aligned} \sum_{i=2}^{n} \sum_{j=1}^{i-1} w_{j,i} \cdot r_{i,j}(H) &= \sum_{i=2}^{n} \sum_{j=1}^{i-1} [1 - r_{j,i}(H)] w_{j,i} \\ &= \sum_{j=2}^{n} \sum_{i=1}^{j-1} [1 - r_{i,j}(H)] w_{i,j} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [1 - r_{i,j}(H)] w_{i,j}, \end{aligned}$$

we get

$$f(H) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} r_{i,j}(H)[w_{j,i} - w_{i,j}] + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j}.$$

We now set, for $1 \leqslant i < j \leqslant n$, $a_{j,i} = w_{j,i} - w_{i,j}$. Hence the expression of f:

$$f(H) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} a_{j,i} \cdot r_{i,j}(H) + C,$$

where $C = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j}$ is a constant.

From now on, we shall use the variables $r_{i,j}(H)$ only for $1 \leqslant i < j \leqslant n$, with their values belonging to $\{0, 1\}$; as H is complete and asymmetric, this set of variables gives the complete description of H. It is easy to check that H is transitive (that is, H is a linear order) if and only if the following inequalities are verified (this can be linked to the integer programming formulation of the feedback arc set problem, see for instance [2,7,14,58,62]):

$$\text{for } 1 \leqslant i < j < k \leqslant n, \quad 0 \leqslant r_{i,j}(H) + r_{j,k}(H) - r_{i,k}(H) \leqslant 1.$$

To sum up the situation, the 0–1 integer programming formulation of our problem (the *primal problem* according to the Lagrangean relaxation terminology) is

$$\text{Minimize } f(H) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} a_{j,i} \cdot r_{i,j}(H) + C$$

under the constraints

$$\begin{cases} \forall\, (i, j, k) & \text{with } 1 \leqslant i < j < k \leqslant n, \ \ 0 \leqslant r_{i,j}(H) + r_{j,k}(H) - r_{i,k}(H) \leqslant 1, \\ \forall\, (i, j) & \text{with } 1 \leqslant i < j \leqslant n, \ \ r_{i,j}(H) \in \{0, 1\}. \end{cases}$$

We denote by f^* the minimum value taken by f over the set of tournaments H satisfying all the constraints, that is, over the set of linear orders.

Relaxing the transitivity constraints gives the Lagrangean function L defined by

$$\begin{aligned} L(H, \Lambda, M) = f(H) &+ \sum_{1 \leqslant i < j < k \leqslant n} \lambda_{i,j,k}[r_{i,j}(H) + r_{j,k}(H) - r_{i,k}(H) - 1] \\ &- \sum_{1 \leqslant i < j < k \leqslant n} \mu_{i,j,k}[r_{i,j}(H) + r_{j,k}(H) - r_{i,k}(H)], \end{aligned}$$

where H is still any tournament defined on X and where the Lagrangean multipliers $\lambda_{i,j,k}$ and $\mu_{i,j,k}$, with $\Lambda = (\lambda_{i,j,k})_{1 \leqslant i < j < k \leqslant n}$ and $M = (\mu_{i,j,k})_{1 \leqslant i < j < k \leqslant n}$, are non-negative real numbers.

It is now very easy to rewrite $L(H, \Lambda, M)$ as

$$L(H, \Lambda, M) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{i,j} \cdot r_{i,j}(H) + C - \sum_{1 \leqslant i < j < k \leqslant n} \lambda_{i,j,k}$$

with

$$b_{i,j} = a_{i,j} + \sum_{k=j+1}^{n} (\lambda_{i,j,k} - \mu_{i,j,k}) + \sum_{k=1}^{i-1} (\lambda_{k,i,j} - \mu_{k,i,j}) - \sum_{k=i+1}^{j-1} (\lambda_{i,k,j} - \mu_{i,k,j})$$

(with the usual understanding that when the lower index of a sum is greater than the upper index, the sum is equal to 0).

For non-negative values of the Lagrangean multipliers, the dual function D is defined as

$$D(\Lambda, M) = \min_{\{H\}} L(H, \Lambda, M),$$

where the minimum is taken over the set of the tournaments H defined on X. The constraints are only that the variables $r_{i,j}(H)$ must belong to $\{0, 1\}$ for $1 \leqslant i < j \leqslant n$. It is very easy to compute $D(\Lambda, M)$ for a given λ and a given M: if $b_{i,j} > 0$, $r_{i,j}(H)$ is set to 0, and otherwise it is set to 1.

Remember that for any non-negative Lagrangean multipliers Λ and M, $D(\Lambda, M)$ is a lower bound of f^*. The dual problem consists in computing the maximum D^* of D on the set of the non-negative Lagrangean multipliers (Λ, M). Trying to solve the dual problem exactly is often too long; so we try to make good choices of the multipliers in order to have a good lower bound for the primal problem; for this, we use a subgradient optimization method (for a global presentation of such a method, see [50] for instance). We start with some arbitrary multipliers (Λ^0, M^0); at iteration p, let $r_{i,j}^p(H)(1 \leqslant i < j \leqslant n)$ be the optimal values of the variables $r_{i,j}(H)$ when we compute $D(\Lambda^p, M^p)$ (where Λ^p

and M^p denote the Lagrangean multipliers at iteration p); to update the multipliers following a subgradient of D at the point (Λ^p, M^p), we set (see [2])

$$\lambda_{i,j,k}^{p+1} = \max\{0, \lambda_{i,j,k}^{p} + \gamma[r_{i,j}^{p}(H) + r_{j,k}^{p}(H) - r_{i,k}^{p}(H) - 1]\},$$
$$\mu_{i,j,k}^{p+1} = \max\{0, \mu_{i,j,k}^{p} - \gamma[r_{i,j}^{p}(H) + r_{j,k}^{p}(H) - r_{i,k}^{p}(H)]\},$$

where γ is a parameter which corresponds to the size of the step. More precisely

$$\gamma = \frac{\rho[B - D(\Lambda^p, M^P)]}{n(n-1)(n-2)},$$

where B is the best (that is, the lowest) known value of the function f and ρ is a parameter which decreases during the process (at each iteration, it is multiplied by a factor θ with $\theta < 1$). The initial choice of ρ is the hard point of the method; if ρ is too high, it is possible that the value of $D(\Lambda^{p+1}, M^{p+1})$ is much smaller than $D(\Lambda^p, M^p)$ and, iteration by iteration, the value of the dual function may go down to $-\infty$: in this case, we say that we get a *drift*. If ρ is too small, it is necessary to perform many iterations and the method is very slow; as this parameter decreases, it is even possible never to move near the maximum of the dual problem. To overcome this difficulty, we choose an initial value of ρ rather big, but if a drift happens, we multiply ρ by a number smaller than 1 (by 0.8 in our software).

To save memory space, we work only with the differences $\lambda_{i,j,k} - \mu_{i,j,k}$ for $1 \leqslant i < j < k \leqslant n$ (see [2] for details).

We may improve Arditti's process by increasing the lower bound of f^* with the following idea. Let Ω be the set of linear orders defined on X. Then, for any given non-negative values of the Lagrangean multipliers Λ and M, we have

$$f^* \leqslant \min_{O \in \Omega} L(O, \Lambda, M),$$

that is,

$$f^* \leqslant \min_{O \in \Omega} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{i,j} \cdot r_{i,j}(O) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j} - \sum_{1 \leqslant i < j < k \leqslant n} \lambda_{i,j,k}.$$

Consider a set $S = S_1 \cup S_2$ of triplets $(i_s, j_s, k_s)_{1 \leqslant s \leqslant |S|}$ such that

(a)
$$\begin{cases} \text{for any } (i_s, j_s, k_s) \in S, & 1 \leqslant i_s < j_s < k_s \leqslant n, \\ \text{if } (i_s, j_s, k_s) \in S_1, & b_{i_s,j_s} > 0, \ b_{j_s,k_s} > 0, \ b_{i_s,k_s} < 0, \\ \text{if } (i_s, j_s, k_s) \in S_2, & b_{i_s,j_s} < 0, \ b_{j_s,k_s} < 0, \ b_{i_s,k_s} > 0, \end{cases}$$

(b) for $(i_s, j_s, k_s) \in S$, let B_s be the set $\{(i_s, j_s), (j_s, k_s), (i_s, k_s)\}$; then, for any s and t with $1 \leqslant s < t \leqslant |S|$, the sets B_s and B_t are disjoint.

The set S can be seen as a collection of arc-disjoint 3-circuits (that is, circuits defined on three vertices and with no arc in common) in the tournament defined on X and oriented by the signs of the numbers b: the arc between i and j with $i < j$ is (i, j) if and only if $b_{i,j} \leqslant 0$. Set

$$\overline{B} = \{(i, j) : 1 \leqslant i < j \leqslant n\} - \bigcup_{s=1}^{|S|} B_S.$$

Then we have

$$L(O, \Lambda, M) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j} - \sum_{1 \leqslant i < j < k \leqslant n} \lambda_{i,j,k} + \sum_{(i,j) \in \overline{B}} b_{i,j} \cdot r_{i,j}(O) + \sum_{s=1}^{|S|} [b_{i_s,j_s} \cdot r_{i_s,j_s}(O) + b_{j_s,k_s} \cdot r_{j_s,k_s}(O) + b_{i_s,k_s} \cdot r_{i_s,k_s}(O)].$$

If O is a linear order, it is impossible to have $r_{i_s,j_s}(O)=0$, $r_{j_s,k_s}(O)=0$ and $r_{i_s,k_s}(O)=1$ simultaneously, as was chosen previously for $(i_s, j_s, k_s) \in S_1$. Similarly, it is impossible to have $r_{i_s,j_s}(O)=1$, $r_{j_s,k_s}(O)=1$ and $r_{i_s,k_s}(O)=0$ simultaneously, as was chosen previously for $(i_s, j_s, k_s) \in S_2$. For $1 \leqslant s \leqslant |S|$, let α_S denote the following quantity:

$$\alpha_S = b_{i_s,j_s} \cdot r_{i_s,j_s}(O) + b_{j_s,k_s} \cdot r_{j_s,k_s}(O) + b_{i_s,k_s} \cdot r_{i_s,k_s}(O).$$

According to the values of $r_{i_s,j_s}(O)$, $r_{j_s,k_s}(O)$ and $r_{i_s,k_s}(O)$, we get the following values of α_S for $(i_s, j_s, k_s) \in S_1$:

- if $r_{i_s,j_s}(O)=1$, $r_{j_s,k_s}(O)=1$ and $r_{i_s,k_s}(O)=1$, then $\alpha_s = b_{i_s,j_s} + b_{j_s,k_s} + b_{i_s,k_s}$;
- if $r_{i_s,j_s}(O)=1$, $r_{j_s,k_s}(O)=1$ and $r_{i_s,k_s}(O)=0$, then $\alpha_s = b_{i_s,j_s} + b_{j_s,k_s}$;
- if $r_{i_s,j_s}(O)=1$, $r_{j_s,k_s}(O)=0$ and $r_{i_s,k_s}(O)=1$, then $\alpha_s = b_{i_s,j_s} + b_{j_s,k_s}$;
- if $r_{i_s,j_s}(O)=1$, $r_{j_s,k_s}(O)=0$ and $r_{i_s,k_s}(O)=0$, then $\alpha_s = b_{i_s,j_s}$;
- if $r_{i_s,j_s}(O)=0$, $r_{j_s,k_s}(O)=1$ and $r_{i_s,k_s}(O)=1$, then $\alpha_s = b_{j_s,k_s} + b_{i_s,k_s}$;
- if $r_{i_s,j_s}(O)=0$, $r_{j_s,k_s}(O)=1$ and $r_{i_s,k_s}(O)=0$, then $\alpha_s = b_{j_s,k_s}$;
- if $r_{i_s,j_s}(O)=0$, $r_{j_s,k_s}(O)=0$ and $r_{i_s,k_s}(O)=0$, then $\alpha_s = 0$.

Thus we get

$$\alpha_s \leqslant \min\{b_{i_s,j_s} + b_{j_s,k_s} + b_{i_s,k_s}, b_{i_s,j_s} + b_{j_s,k_s}, b_{i_s,j_s} + b_{i_s,k_s}, b_{i_s,j_s}, b_{j_s,k_s} + b_{i_s,k_s}, b_{j_s,k_s}, 0\}$$

As, still for $(i_s, j_s, k_s) \in S_1$, b_{i_s,j_s} and b_{j_s,k_s} are positive and b_{i_s,k_s} is negative, we have

$$\alpha_s \leqslant \min\{b_{i_s,j_s} + b_{i_s,k_s}, b_{j_s,k_s} + b_{i_s,k_s}, 0\},$$

that is also

$$\alpha_s \leqslant b_{i_s,k_s} + \min\{b_{i_s,j_s}, b_{j_s,k_s}, -b_{i_s,k_s}\}$$

instead of $\alpha_s \leqslant b_{i_s,k_s}$ as described previously (this lower bound corresponding to $r_{i_s,j_s}(O)=0$, $r_{j_s,k_s}(O)=0$ and $r_{i_s,k_s}(O)=1$). It means that the contribution of $(i_s, j_s, k_s) \in S_1$ to the computation of $L(O, \Lambda, M)$ may be increased by a "gain" $g(s)$ defined by

$$\begin{aligned} g(s) &= \min\{b_{i_s,j_s}, b_{j_s,k_s}, -b_{i_s,k_s}\} \\ &= \min\{|b_{i_s,j_s}|, |b_{j_s,k_s}|, |b_{i_s,k_s}|\}. \end{aligned}$$

Similarly, for $(i_s, j_s, k_s) \in S_2$, we get

$$\alpha_S \leqslant b_{i_s,j_s} + b_{j_s,k_s} + \min\{-b_{i_s,j_s}, -b_{j_s,k_s}, b_{i_s,k_s}\}$$

instead of $\alpha_S \leqslant b_{i_s,j_s} + b_{j_s,k_s}$ as described previously (this lower bound corresponding to $r_{i_s,j_s}(O)=1$, $r_{j_s,k_s}(O)=1$ and $r_{i_s,k_s}(O)=0$). It means that the contribution of $(i_S, j_S, k_S) \in S_2$ to the computation of $L(O, \Lambda, M)$ may be increased by a "gain" $g(s)$ defined by

$$\begin{aligned} g(s) &= \min\{-b_{i_s,j_s}, -b_{j_s,k_s}, b_{i_s,k_s}\} \\ &= \min\{|b_{i_s,j_s}|, |b_{j_s,k_s}|, |b_{i_s,k_s}|\}. \end{aligned}$$

Finally, let $g(S)$ be the quantity

$$g(S) = \sum_{(i_s,j_s,k_s) \in S} \min\{|b_{i_s,j_s}|, |b_{j_s,k_s}|, |b_{i_s,k_s}|\}.$$

What is done above shows the inequality

$$\min_{O \in \Omega} L(O, \Lambda, M) \leqslant D^* + g(S).$$

On the other hand, we have $f^* \leqslant \min_{O \in \Omega} L(O, \Lambda, M)$ and moreover the values taken by the objective function f are integers; hence we get

$$f^* \geqslant \lceil D^* + g(S) \rceil$$

for any set S satisfying the above conditions. This means that we may improve the lower bound D^* found by Arditti by adding $g(S)$ to it. Then a heuristic is applied to find a "good" set S in order to compute the best possible $g(S)$ in a reasonable CPU time. The choice for this heuristic is a simple greedy heuristic: we first compute a triplet (of type S_1 or S_2) with a maximum gain, then we compute another triplet with a maximum gain among the triplets disjoint with respect to the one already computed, and so on. We shall say that we use the *gain option* when we increase the lower bound D^* of f^* with the help of this gain $g(S)$ for some family S of disjoint triplets.

3. Application of a noising method to the linear ordering problem

To obtain a first bound when the branch-and-bound process is applied to the linear ordering problem, and also to get a good approximate solution for big problems, we apply a noising method, a recent combinatorial optimization metaheuristic first designed in 1993 [19].

3.1. Principles of the noising methods

As other metaheuristics (see [1,26,47,54] for references), the noising methods are not designed to solve only one specific problem, but more generally combinatorial optimization problems. Such a problem can be described as follows:

$$\min_{z \in Z} u(z),$$

where Z is a finite set, of which the elements are called *solutions*, and where $u(z)$ gives the value of the solution $z \in Z$ with respect to the objective function u.

The noising methods are based on elementary (or local) transformations. An elementary transformation is an operation which, when applied to a solution $z \in Z$, changes z into another solution z' of Z by modifying one (or some) feature(s) of z without changing its global structure. For instance, if z is a binary string, an elementary transformation may consist in changing one bit into its complement. The new solution z' is called a *neighbour* of z. More generally, the neighbourhood $\Gamma(z)$ of z is the set of solutions that we can get by applying the elementary transformation to z. For instance, if z is still a binary string of q bits and if the elementary transformation is the one evocated previously, $\Gamma(z)$ is a set of q binary strings: the q binary strings got by changing one of the q bits of z. (See [1,54] for more details about these basic definitions.)

Thanks to a given elementary transformation (or thanks to a given neighbourhood), we may design an *iterative improvement method*, also called a *descent* for a minimization problem. In a descent, we start from an initial solution z_0 (for example randomly generated) and we generate η new solutions $z_i (1 \leqslant i \leqslant \eta)$ with the following properties:

1. for $1 \leqslant i \leqslant \eta$, $\quad z_i \in \Gamma(z_{i-1})$;
2. for $1 \leqslant i \leqslant \eta$, $\quad u(z_i) < u(z_{i-1})$;
3. for $z \in \Gamma(z_\eta)$, $\quad u(z) \leqslant u(z_\eta)$.

Property 3 means that a descent stops when a local (with respect to the adopted elementary transformation or to the adopted neighbourhood) minimum has been found.

The noising methods are based on the same principles but, instead of the genuine function u to minimize, we consider that u has been perturbed by *noises*. Thus, as for a descent, the noising methods apply elementary transformations; but to know whether such a transformation is accepted or not, we take the noises into account. One possibility (see [21] for other possibilities) consists in adding these noises to the variation of u: when a neighbour z' is tried instead of the current solution z, the genuine variation $u(z') - u(z)$ is not taken into account, but a noised variation $u(z') - u(z) + r$, where r is the random noise drawn into an interval (for instance $[-R, R]$ if R denotes the current noise-rate) with a given probability distribution (for instance a uniform law). This involves that a "bad" transformation (that is, a transformation leading to an increase of u) can be accepted (as in simulated annealing for instance), but also that a "good" one (that is, a transformation leading to a decrease of u) can be rejected because of the noises. In order to get back the genuine function u to minimize at the end of the process, the range R of the noises decreases during the run of the method, typically down to 0 (but it is often possible to stop before): then there is no added noise and we deal with u itself.

Thus, it is necessary to specify different components to get a noising scheme: the way to explore the neighbourhood, how to choose the initial (and maximum) value of the noise-rate R, how to make R decrease, how to choose the noises, and so on (see [21] for a survey on the noising methods). It is what we do below for the application of a noising method to our linear ordering problem.

3.2. Application of a noising method to the linear ordering problem

To initialize the metaheuristic, we start with a linear order randomly chosen. The neighbourhood is explored as follows: we iteratively compute the best (with respect to the current linear order) place for x_1, then for $x_2, \ldots$, and so on until x_n, and after this, once again for x_1 and so on. If the noise is equal to 0, to look for the best place for a vertex x consists in shifting x in the current order to put x at the rank which minimizes the objective function f while the rest of the current order (the relative ranking of the other vertices) remains the same. It means that, if the current linear order is $O = x_{i_1} > \cdots > x_{i_{p-1}} > x_{i_p} > x_{i_{p+1}} > \cdots > x_{i_n}$ and if x is at the pth place in O (thus $x = x_{i_p}$), we try to put x before x_{i_1} to get the order, $x_{i_p} > x_{i_1} > \cdots > x_{i_{p-1}} > x_{i_{p+1}} > \cdots > x_{i_n}$, or just after x_{i_j} to get the orders for $x_{i_1} > \cdots > x_{i_j} > x_{i_p} > x_{i_{j+1}} > \cdots > x_{i_{p-1}} > x_{i_{p+1}} > \cdots > x_{i_n}$ for $1 \leqslant j < p$ if $1 \leqslant j < p$, or the orders $x_{i_1} > \cdots > x_{i_{p-1}} > x_{i_{p+1}} > \cdots > x_{i_j} > x_{i_p} > x_{i_{j+1}} > \cdots > x_{i_n}$ if $p < j < n$, or the order $x_{i_1} > \cdots > x_{i_{p-1}} > x_{i_{p+1}} > \cdots > x_{i_n} > x_{i_p}$ (for $j = n$). When there is a noise, let us denote by R the current noise-rate and by W the maximum weight of the tournament T. For each possible place j of $x = x_{i_p}$ (defined as above, with $j = 0$ if x is put before x_{i_1}), let $\Delta f(p, j)$ be the variation of the objective function f when x is moved from its current place p to this possible place j. Instead of $\Delta f(p, j)$, we associate with j a noised variation $\Delta f_{\text{noised}}(p, j)$ defined by:

$$\Delta f_{\text{noised}}(p, j) = \Delta f(p, j) + R \times W \times \sum_{k=1}^{|j-p|} \xi(k)$$

where $\xi(k)$ is randomly chosen with a uniform law of probability on the interval $[-1, +1]$.

We also tried another noising-scheme, giving the following value to $\Delta f_{\text{noised}}(p, j)$ for the new place j:

$$\Delta f_{\text{noised}}(p, j) = \Delta f(p, j) + R \times W \times \ln(\xi)$$

where ξ is here uniformly chosen in the interval $]0, 1[$ (this distribution is the same as the one used in simulated annealing; see [21] for details); we obtained results with nearly the same quality, rather a bit worse, so we kept the first solution (see also [20]).

After having computed the values of anyone of the n possible places for x, we put x at its best place, that is the place involving the smallest variation Δf_{noised}.

As the process goes on, R arithmetically decreases from an initial value down to 0. Nevertheless, from time to time, we insert a descent (which corresponds to the noising method with a noise-rate equal to 0); a descent is completed when every vertex is at its best place when we try to shift it while the other vertices keep their current places in the order. After a descent, the parameter R is given back the value it had before the unnoised descent. Moreover, also from time to time, we change the current linear order, replacing it by the best linear order computed since the beginning of the noising method (see [21] for details about these variants). The solution returned at the end of the process is the best linear order computed during the whole process.

As it is necessary to tune some parameters (such as the initial value of the noise-rate) in order to apply the noising method, we apply in fact a self-tuned noising method (see [22]), so that the user has nothing to tune. The automatic tunings for our problem are depicted in Section 5.

4. The branch-and-bound algorithm

The linear ordering problem has been extensively studied and several branch-and-bound methods have been designed for this problem (see for example [11,13,15,17,25,27–29,31–36,38–41,45,46,48,52,53,55,56,59–62]). For instance, Korte and Oberhofer [45,46] solved random tournaments (with the same probability for the two orientations of each arc) with 13 vertices; according to Kaas [41], Lenstra Jr. [48] improved their software and solved problems with up to 17 vertices; Wessels [60] increased this number to 25 in 1981; the experiments of Kaas [41] deal with tournaments with 34 vertices coming from life problems or random tournaments with 25 vertices; in the meanwhile, with the help

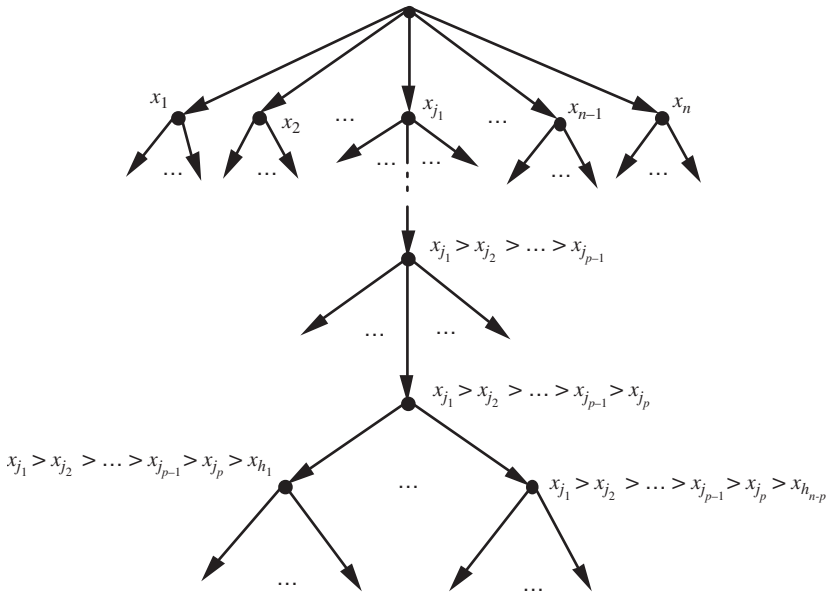

Fig. 1. Structure of the BB-search-tree.

of linear programming, Boenchendorf on one hand [12] and Marcotorchino and Michaud on the other [49] solved life problems with up to 30 vertices for Boenchendorf and up to 72 vertices for Marcotorchino and Michaud (in fact, there was only one case with 72 vertices, the others have at most 36 vertices); we may also mention the works of Grötschel, Jünger and Reinelt who applied their polyhedral methods to life problems with up to 71 vertices [24] or to tournaments of which the weights simulate life data with up to 80 vertices ([28,30]; see also [29,40,55]).

Our branch-and-bound method is based on the branch-and-bound designed in [5], but the expanding strategy applied here is a depth-first one instead of the best-first one adopted in [5]. We have improved this basic BB-method by means of several theoretical or practical results. They are summed up below.

At the beginning, we apply the noising method detailed above to get a upper bound B and a linear order which gives the current solution and, at the beginning, the best order O^* computed since the beginning; thus, we have $f(O^*) = B$ before starting the BB-method. This relation will remain true when B and O^* change during the BB-method.

A node N of the BB-tree contains all the linear orders beginning with a given sequence $x_{i_1} > x_{i_2} > \cdots > x_{i_{p-1}} > x_{i_p}$: this sequence is called a *beginning section*. The "parent" of N in the BB-tree is the node containing all the linear orders beginning with $x_{i_1} > x_{i_2} > \cdots > x_{i_{p-1}}$; a "child" of N, if it exists (N may have between 0 and $n - p$ such "children"), is a node which contains the linear orders beginning with $x_{i_1} > \cdots > x_{i_p} > x$, for some vertex $x \notin \{x_{i_1}, \ldots, x_{i_p}\}$, as shown in Fig. 1.

Let us define the value $V(BS)$ of a beginning section $BS = x_{i_1} > \cdots > x_{i_p}$ by

$$V(BS) = \sum_{\substack{1 \leqslant h < k \leqslant p \\ (x_{i_k}, x_{i_h}) \in E}} w_{i_k, i_h} + \sum_{k=1}^{p} \sum_{\substack{(x_j, x_{i_k}) \in E \\ x_j \notin \{x_1, \ldots, x_p\}}} w_{j, i_k}.$$

In other words, the value of the beginning section BS is the sum of the weights of the arcs that must be reversed to get any linear order beginning with BS. Then, for any such linear order O, we have $f(O) \geqslant V(BS)$: $V(BS)$ is a lower bound of the minimum value taken by f over the set of linear orders beginning with BS. It is this value which is considered for designing the evaluation function in [5]. Several improvements of this evaluation function are proposed in [18] in order to anticipate on what will be necessary to reverse in the subtournament induced by the vertices not appearing inside BS. Sections 4.1–4.6 describe the components of the BB-method (from our experimentations, it appears that the other improvements proposed in [18] give less good results).

To study a node N of the BB-tree consists in finding the best order with respect to the objective function f beginning with the beginning section BS associated with N. In the rest of Section 4, we describe how we study a node and we detail below the successive procedures applied when a node is studied. For this, we denote by S the set of vertices of the beginning section BS associated with the studied node N, by V the value $V(BS)$ of BS, and by OS (for "out of section") the set of vertices which do not belong to S.

4.1. The depth

If $p = n - 1$, then the studied node contains only one order O and V is equal to $f(O)$. So we compare V to the current bound B: if $V < B$, then O^* is updated with O, and B with V. In this case, the study of the node is finished. Otherwise, we go on with the test described in 4.2.

4.2. The tree of beginning sections

The following idea of keeping the sets of vertices of all the beginning sections already computed in a tree lexicographically organized was suggested by Guénoche in [34]. For example, the beginning sections $x_4 > x_7 > x_5 > x_2$ and $x_5 > x_4 > x_2 > x_7$ correspond to the same set of vertices $\{x_2, x_4, x_5, x_7\}$. For each set S' of vertices such that a beginning section defined on S' has been generated, we store the best value of the already computed beginning sections defined on S'. When we study the node N, we check if the set S associated with N is already in the tree. If it is the case, we compare the stored value and V: if the stored value is smaller than or equal to V, the study of N is finished (N does not contain any order better than the current best order O^*); otherwise, we keep V instead of the stored value associated with S and we go on with the test described in Section 4.3. If S is not in the beginning-sections-tree, we create a new branch in this tree to insert S with V as its stored value, and we go on with the procedure in Section 4.3. In fact, another value will be also stored in this tree at the place associated with S: it will be the value found by the Lagrangean relaxation when applied (if applied) to the tournament induced by OS (see Section 4.5).

If the beginning-sections-tree takes too much memory space, this procedure (Section 4.2) is still applied and the values of the beginning sections already stored are still updated, but the beginning-sections-tree is no more expanded during its updating.

In the section devoted to results (Section 6), *BegSec* will count the number of nodes of the BB-tree eliminated according to this principle.

4.3. The parameter σ

This procedure is applied only if all the weights are equal to 1, that is only in the case of the problem of Slater.

For a tournament $T = (X, E)$, we first compute the out-degrees of the vertices and we sort them in order to get an increasing sequence $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_n$; the parameter $\sigma(T)$ is defined in [16] by

$$\sigma(T) = \frac{1}{2} \sum_{j=1}^{n} |s_j - j + 1|.$$

It is shown in [16] that Slater's index $i(T)$ of T is greater than or equal to $\sigma(T) : \forall T,\ \sigma(T) \leqslant i(T)$. So, for any beginning section S, we compute the value $\sigma(T_{OS})$ of the parameter σ for the tournament T_{OS} induced by OS. Then, to get a linear order defined on X and beginning with BS, it is necessary to reverse at least $\sigma(T_{OS}) + V$ arcs (exactly V to get BS and at least $\sigma(T_{OS})$ to make T_{OS} transitive). So, if $\sigma(T_{OS}) + V$ is greater than or equal to the bound B, the study of the current node N is finished: N cannot contain an order better than O^*; otherwise, we continue with the test 4.4.

4.4. A descent

For any beginning section S, we apply a descent (see Section 3) to try to solve the linear ordering problem for the tournament T_{OS} induced by the vertices which are in OS; let v be the value computed by this descent. If $v + V(S)$ is

less than B, we update B and O^* (B is replaced by $v + V(S)$ and O^* by the concatenation of the beginning section BS with the linear order defined on OS found by the descent). A more sophisticated heuristic could be applied instead of a descent. But, according to our experiments, the bound B found by the noising method is almost always the exact value (see the end of Section 6), even if, of course, we do not know this yet; so, applying a heuristic more sophisticated than a descent usually consumes more CPU time than it saves (it is not the case with a descent, because a descent needs very little CPU time).

4.5. Evaluation provided by the Lagrangean relaxation

If there is enough memory space for the Lagrangean multipliers, we use the Lagrangean relaxation to improve the evaluation $V(S)$ of the considered beginning section S. (In fact, to save memory space, we do not keep the multipliers Λ and M in memory, but only their differences $\Lambda - M$; see [2]). More precisely, we apply the relaxation method described in Section 2 to the tournament T_{OS} induced by the vertices which are not in S (if there is not enough memory space, we definitively cancel the use of the Lagrangean relaxation). Remember that any value of D computed during the relaxation (more precisely, the lowest integer greater than or equal to any value of the dual function D plus the gain g if the gain option is activated) gives a lower bound of f with respect to T_{OS}. So, during the method, if the sum of $V(S)$ and a value provided by the relaxation is greater than or equal to B, we stop the relaxation and the study of the current node is finished.

In order to save time, we initialized the values of the Lagrangean multipliers by the ones obtained at the end of the previous relaxation (for the first relaxation, when we are at the root of the tree, the multipliers are initialized to 0); clearly, during a relaxation applied to the vertices in OS, only the multipliers $\lambda_{i,j,k}$ and $\mu_{i,j,k}$ with the three vertices x_i, x_j, x_k in OS are used and updated. When a relaxation begins, it is almost sure that these previous values are better than 0.

We had to decide whether it is worth trying to cut a branch of the BB-tree by means of the gain option. It is not obvious, because computing the gain may take much CPU time. We observed that it is surely better to use the gain option at the first iteration of each relaxation (an iteration being the computation of one value of the dual function and of the new values of the Lagrangean multipliers), but not to perform it for the other iterations (for them, the gain is usually too small to provide significant improvements). Moreover, the gain option also requires memory space; if there is no more space for it, we definitively stop the use of this option.

For the first relaxation, at the root, it can be a good idea to perform a great amount of iterations of the relaxation method. It gives a chance to cut the branch-and-bound process at the root, and it tunes the Lagrangean multipliers for the next relaxations, if any. For the other relaxations, it is better to do only a small number of iterations; with a great number of iterations, maybe there would be a smaller number of nodes studied during the BB-method, but the CPU time spent for each node would be bigger. More details on the tuning of these parameters are given in Section 5.

At the end of the relaxation, we save the best value obtained for the dual function corresponding to the tournament induced by the vertices belonging to OS in the beginning-sections-tree detailed in Section 4.2. We will later use this value before applying a relaxation to a node having the same set S of vertices in its beginning section: perhaps this value will be enough to show that this new node cannot contain an order better than O^*, which would avoid to perform the relaxation.

If the relaxation does not show that the current node can be pruned, we keep on with the branching described in the next section.

In Section 6, *relax* will count the number of nodes eliminated according to the relaxation principle.

4.6. Branching

For each vertex x of OS, we do what is explained below. Let $x_{i_1} > x_{i_2} > \cdots > x_{i_p}$ be the current beginning section defined on S.

First, we apply some tests in order to prove that it is useless to study the new beginning section obtained by adding x just after the current beginning section, that is the beginning section $x_{i_1} > x_{i_2} > \cdots x_{i_p} > x$. In these tests, two types of arguments allow us to reject the new beginning section:

(1) sometimes, it is possible to show that, for structural reasons, this new beginning section is incompatible with any median order (there is no median order beginning with this beginning section);

(2) sometimes, it is possible to show that for each order beginning with the new beginning section, there is another one with the same value, but with a beginning section "smaller" with respect to the lexicographic order: in this case, as this "smaller" beginning section will be met or has already been met elsewhere in the BB-search-tree, we do not study the new beginning section. It is the only case for which we do not study a node of the BB-tree which can contain an order better than O^*: only the "smallest" (with respect to the lexicographic order) beginning section defined on the same vertices will not be rejected with this argument. Notice that the procedure of the beginning-sections-tree explained in Section 4.2 also avoids to examine beginning sections of the same value defined on a same set of vertices, but we kept the lexicographic arguments developed here because they are faster to test and because they can be applied even if the procedure of Section 4.2 is stopped, which happens when there is not enough memory space for the tree of beginning sections.

More precisely, when we deal with the beginning section $BS = x_{i_1} > x_{i_2} > \cdots > x_{i_p} > x$ obtained by adding x to $x_{i_1} > x_{i_2} > \cdots > x_{i_p}$, we consider the beginning sections that we can get by moving, inside BS, an "interval" of BS including x, that is, the beginning sections in the following form:

$$x_{i_l} > x_{i_2} > \cdots > x_{i_{j-1}} > x_{i_{k+1}} > x_{i_{k+2}} > \cdots > x_{i_p} > x > x_{ij} > x_{i_{j+1}} > \cdots > x_{i_k},$$

where $1 \leqslant j \leqslant k \leqslant p$. If any of these beginning sections is better than BS or has the same value as BS but is smaller than BS with respect to the lexicographic order, then we do not keep BS. In Section 6, *lex* will count the number of nodes eliminated from the BB-tree because such a move leads to a beginning section as good as BS with respect to the weights of the arcs that it is necessary to reverse to get it, but smaller than BS with respect to the lexicographic order. *Smoves* will count the number of nodes eliminated from the BB-tree because such a move leads to a beginning section (strictly) better than BS, except for $j = k = p$. When $k = j$ (we shift x_{i_j} behind x) or when $k = p$ (we shift x before x_{i_j}), the comparison corresponds to what happens when we move only one vertex. The special case $k = j = p$ corresponds to a swap between x and x_{i_p}; it is linked to a result due to Remage and Thompson [56] for unweighted tournaments T and which can be generalized to weighted tournaments (see [18]), saying that if $x_{i_1} > x_{i_2} > \cdots > x_{i_n}$ is a Slater order of the unweighted tournament T, then $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ is a Hamiltonian path of T; what we get for $k = j = p$ is in fact a generalization of this result (see [18]), since the lexicographic argument applied to the beginning sections allows us to consider weighted tournaments, even with weights equal to 0 (what happens for the Feedback Arc Set problem). For this reason, when there is a gain in the swap of x and x_{i_p}, we will not increase the counter *Smoves*, but a new one called *ham*.

We also compute the variation of the objective function when, from an order beginning with the new beginning section BS, we take an interval $x_{i_j} > x_{I_{j+1}} > \cdots x_{i_p} > x$ $(1 \leqslant j \leqslant p)$ and we shift it at the end of the order, after the vertices of OS: if the objective function decreases, we do not keep BS; we will count this kind of cuts in *OSmoves* in Section 6. This test implements a generalization (see [18]) of a result due to Bermond [10], saying that, if x is the first vertex of a Slater order of an unweighted tournament T, then the out-degree of x in T is at least equal to $(n-1)/2$.

If BS is not eliminated by these tests, we create a new node in the BB-tree and we recursively study it by applying the rules depicted in this section, until all the branches are cut or studied.

5. The choice of parameters

The noising method and the Lagrangean relaxation involve the tuning of parameters. Some of them have been fixed as constants in our program; the others are computed as functions of some characteristics of the tournament T: its order n, its weights and its *transitivity index* $\tau(T)$ of Kendall and Babington Smith [44]; this transitivity index is defined by

$$\tau(T) = 1 - \frac{\text{numer of 3-circuits of } T}{\text{maximum number of 3-circuits of a tournament of order } n}.$$

The number of 3-circuits of T is easily computed from the sequence of the out-degrees of the vertices of T. More precisely, if s_j denotes the out-degree of x_j $(1 \leqslant j \leqslant n)$, then (see [51]) the number of 3-circuits of T is equal to

$$\frac{n(n-1)(n-2)}{6} - \sum_{j=1}^{n} \frac{s_j(s_j-1)}{2}$$

while the maximum of this number is equal to $(n^3 - n)/24$ if n is odd or to $(n^3 - 4n)/24$ if n is even. The transitivity index of a transitive tournament is equal to 1; a tournament with all the out-degrees equal to $(n-1)/2$ if n is odd or with $n/2$ out-degrees equal to $(n/2)$ and $(n/2)$ out-degrees equal to $(n/2)-1$ if n is even has a transitive index equal to 0. Though two tournaments with the same transitivity index may be more or less difficult to solve, this quantity gives an indication upon the difficulty: broadly speaking, the lowest the transitivity index, the most difficult.

We tried to choose good constants and good functions in order to get efficient parameters in average. The following tunings arise from our experiments.

For the noising method, we applied the self-tuning of the parameters described in [22]:

- the maximum noise-rate $R_{\max}$ is automatically computed from the characteristics of T: n, the weights of T, $\tau(T)$;
- the rate of noise at the end of the method has been fixed at 0;
- the CPU time devoted to the noising method is equal to $10^{-4}n^3(1+\varepsilon-\tau(T))$ s where $\varepsilon = 10^{-5}$ is here just for the case of a transitive tournament (notice that the size of the neighbourhood induced by the shifting elementary transformation described in Section 3 is about n^2); to be more specific, it means that for the tournament T called Judges100 in Section 6 with $\tau(T) = 0.8275$ and $n = 100$, the noising method spends about 17 s, that is less than 0.7% of the total CPU time necessary to find a median order (see Section 6);
- the other parameters are chosen according to the classic patterns described in [21,22].

For the relaxation method, we must distinguish between two cases: the first relaxation applied at the root of the BB-search-tree, and the other relaxations.

For both cases, we consider that we have a drift if the value of the Lagrangean function at a given iteration is less than its value in the previous iteration multiplied by a constant fixed at 0.98. When a drift occurs, the current value of the parameter ρ used in Section 2 to update the Lagrangean multipliers from an iteration to the next one is multiplied by a constant equal to 0.8.

For the first relaxation:

- the initial value ρ^1_{init} of the parameter ρ is chosen as an increasing, affine by intervals function of the transitive index $\tau(T)$ of the tournament T that we consider; more precisely, we chose:

$$\rho^1_{\text{init}} = \begin{cases} 6 & \text{if } \tau(T) = 0, \\ 66 & \text{if } \tau(T) = 0.5, \\ 180 & \text{if } \tau(T) = 0.8, \\ 480 & \text{if } \tau(T) = 0.9, \\ 1200 & \text{if } \tau(T) = 1 \end{cases}$$

  and between the values 0, 0.5, 0.8, 0.9 and 1 of $\tau(T)$, the function ρ^1_{init} is affine;
- the number of iterations I_1 of the first relaxation is a constant equal to 1000;
- the decreasing factor θ is equal to $1-(0.1/I_1)$, that is, to 0.9999.

For the other relaxations:

- as long as we have no drift, the initial value ρ^2_{init} of the parameter ρ is equal to $\rho^1_{\text{init}}/2$; when there is a drift, ρ^2_{init} is multiplied by the constant 0.9;
- the number of iterations is a constant I_2 equal to 20;
- the decreasing factor θ is equal to $1-(0.1/I_2)$, that is, to 0.995.

6. Results

We give two kinds of results. With the first kind, we study the impact of the various components of the branch-and-bound method; with the second kind, we give the computing time for different problems, in order to show what it is possible to do with our method. We report only a part of our experiments; the others lead to the same qualitative

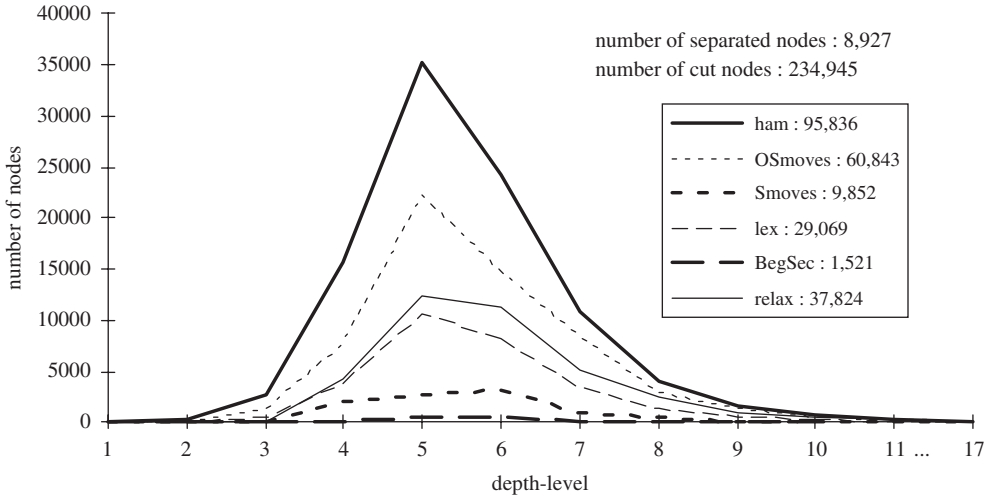

Fig. 2. Numbers of nodes cut for Slater32.

conclusions. The experiments were done on a workstation SUN Sparc. The times given below are CPU times, expressed in seconds.

6.1. The different components of the branch-and-bound method

The experiments that we report here are done on three graphs (but other ones lead to the same qualitative results). Their names, orders, maximum weights, transitivity indices and the CPU times (in s) necessary to our software to solve these problems are given in the following table:

Name	Order	Max weight	τ	CPU time (s)
Slater32	32	1	0.0875	509
Median39	39	10	0.0785	2208
Judges100	100	44	0.8275	2451

For Slater32, for each pair of vertices $\{x_i, x_j\}$, with $1 \leqslant i < j \leqslant n$, we draw the orientation of the arc between x_i and x_j with a probability equal to 0.5 for each orientation (this distribution leads usually to difficult tournaments, which is confirmed here by the very low value of the transitivity index). All the weights are equal to 1.

For Median39, for each pair of vertices $\{x_i, x_j\}$, with $1 \leqslant i < j \leqslant n$, we draw the orientation of the arc between x_i and x_j with a probability equal to 0.5 for each orientation. Then, for each arc, its weight is randomly chosen, with a uniform law of probability, between 1 and 10.

For Judges100, we imagine that $n = 100$ candidates (the vertices of the tournament Judges100) must be ranked by 50 "judges". To do this, each judge decides, for each pair $\{x_i, x_j\}$ of candidates with $1 \leqslant i < j \leqslant n$, whether he or she prefers x_i to x_j with a probability equal to $0.5 + 0.35(j - i)/(n - 1)$; if he or she does not prefer x_i to x_j, he or she prefers x_j to x_i. Then we defined the quantity $w_{i,j}$ by

$w_{i,j}$ = number of judges preferring x_i to x_j-number of judges preferring x_j to x_i if $i \leqslant j$ and $w_{i,i} = 0$ for $1 \leqslant i \leqslant n$. The arc between x_i and x_j is (x_i, x_j) if $w_{i,j} > 0$ or if $w_{i,j} = 0$ and $i < j$ (otherwise, the arc is oriented from x_j to x_i in order to get a tournament and is weighted by $-w_{i,j}$).

For each one of these three graphs, we applied our method and, to measure the efficiency of each component, we computed the numbers of nodes cut by each component of our method at each depth-level in the BB-search-tree. We obtained Figs. 2–4; in their legends, we specify the total number of nodes cut by each component. There is no curve for the parameter σ (which is only used for unweighted tournaments) because this component cuts no branch at all in the case of Slater32 (see below).

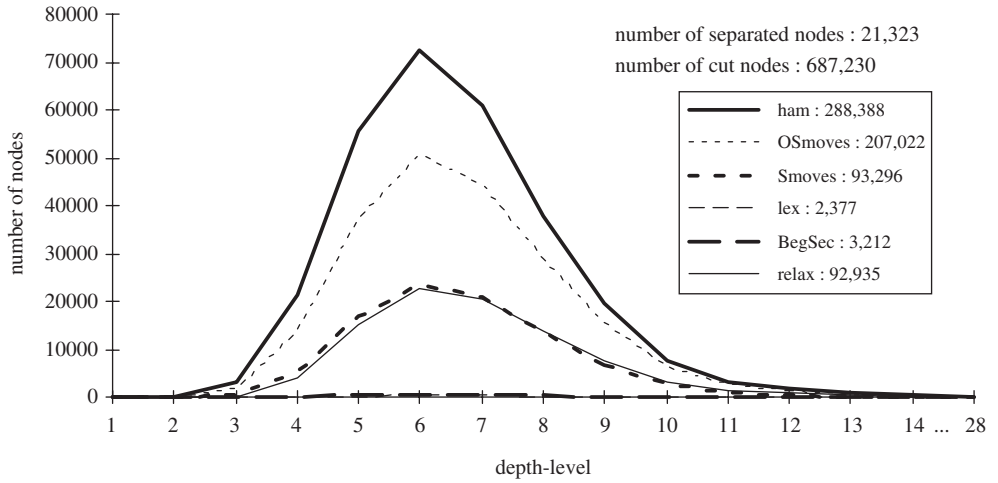

Fig. 3. Numbers of nodes cut for Median39.

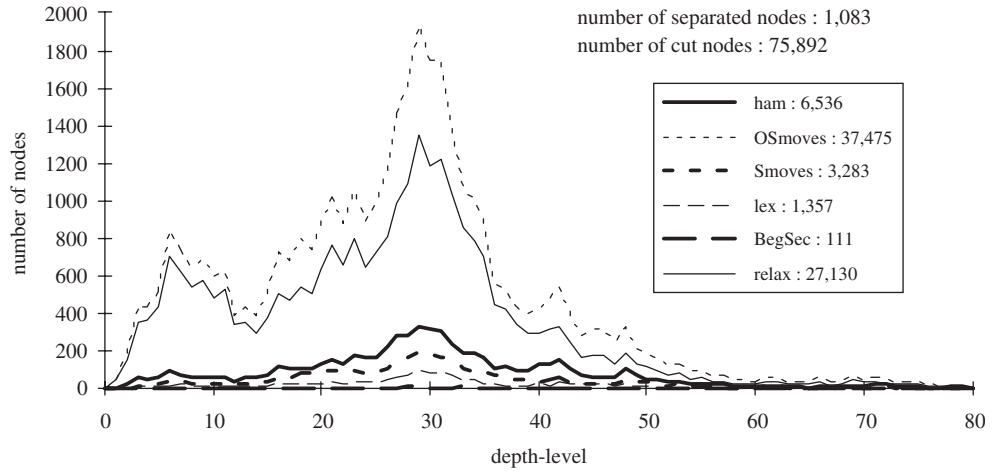

Fig. 4. Numbers of nodes cut for Judges100.

When we analyse the above curves, it is very important to remember the order in which we apply the six components when we try to cut a branch. The first one is the principle of the Hamiltonian path (see Section 4.6; it corresponds with *ham*). Then there are the moves of which the impacts are given by *OSmoves*, *Smoves* and *lex* (see Section 4.6; the impacts of the moves corresponding with *lex* and *Smoves* are measured simultaneously since these tests are done on the same intervals). The use of the tree of the beginning sections (see Section 4.2), of which the effects are measured by *BegSec*, comes after. The last component to be applied is the Lagrangean relaxation (corresponding with *relax*; see Sections 2 and 4.5). If we change this order, the results can be quite different. For example, if *BegSec* is performed before *Smoves*, it cuts much more branches because these two components cut partly the same branches: they avoid to study some beginning sections which are worse than other beginning sections defined on the same vertices.

From these curves, it appears that, in our experiments, the most efficient components are *ham*, *OSmoves* and *relax*; *Smoves* still brings significant improvements for tournaments of the same type as Median39 and *lex* for unweighted tournaments like Slater32; *BegSec* is seldom used (but this is partly due to its place in the software). Nevertheless, it happens that one component cuts few branches, but whose examination would take a very long time otherwise.

The parameter σ did not cut any branch in many cases among the unweighted tournaments that we considered, except for almost transitive tournaments and for a special tournament, not a random one, having 200 vertices and a small index

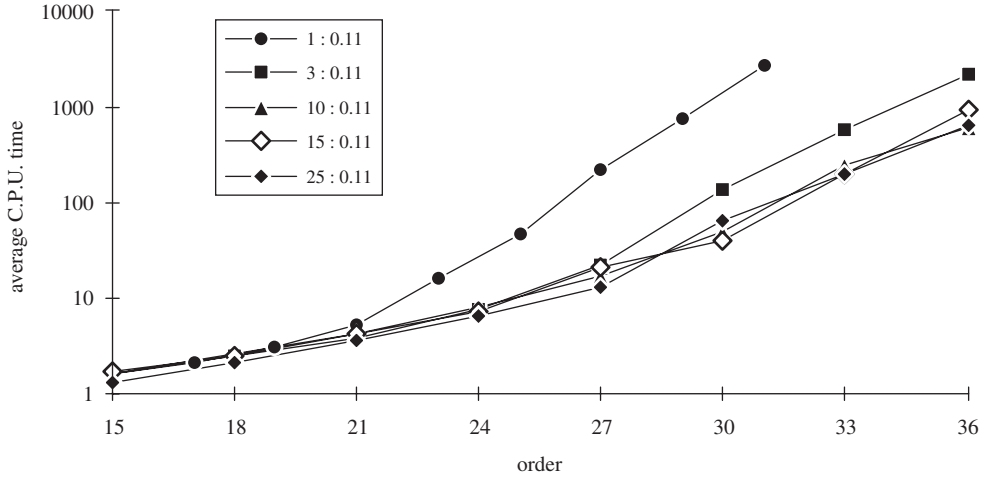

Fig. 5. Time to solve 0.5–0.5 problems.

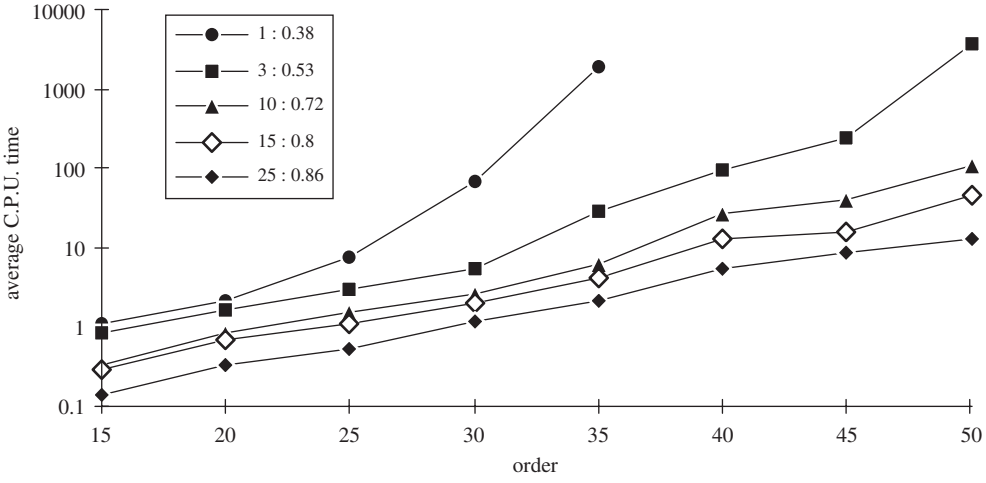

Fig. 6. Time to solve 0.5–1 problems.

of transitivity; in this case, this component cuts the BB-tree at the root and, without it, this problem would be very difficult to solve. So, for such special cases, we chose to keep this component, which consumes only a little time, in our software.

6.2. Computing time

In this section, we give the results got by the application of our software to 5790 tournaments with different characteristics. More precisely, for each set of characteristics (see below), we generate 30 random tournaments with these characteristics and we compute the average CPU time (in seconds) necessary to solve them. Each average CPU time is represented below by a point on a curve in Figs. 5–8 (thus we have 193 points below, corresponding with the 193 sets of 30 tournaments each). The scale of the vertical axis, giving the average CPU time, is logarithmic. The 193 sets of tournaments are built as follows.

All the studied tournaments come from the aggregation of binary relations given by a certain number of virtual judges.

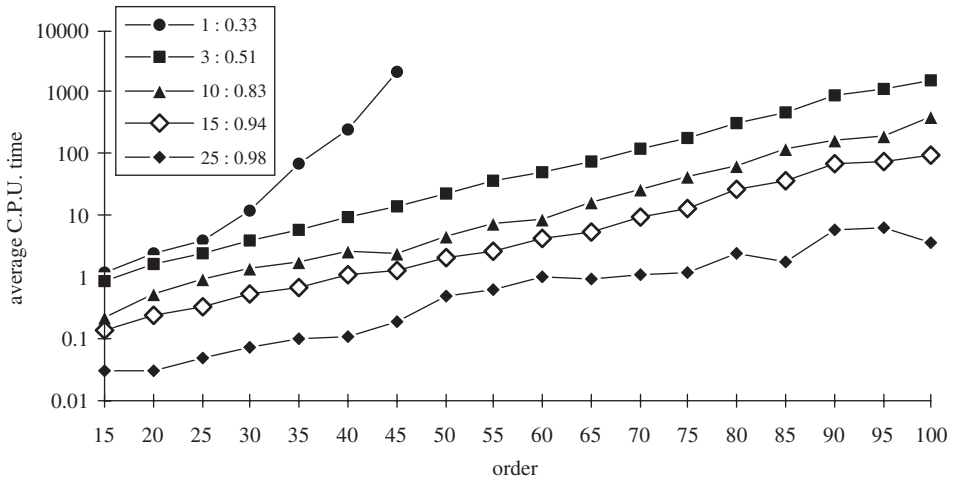

Fig. 7. Time to solve 0.75–0.75 problems.

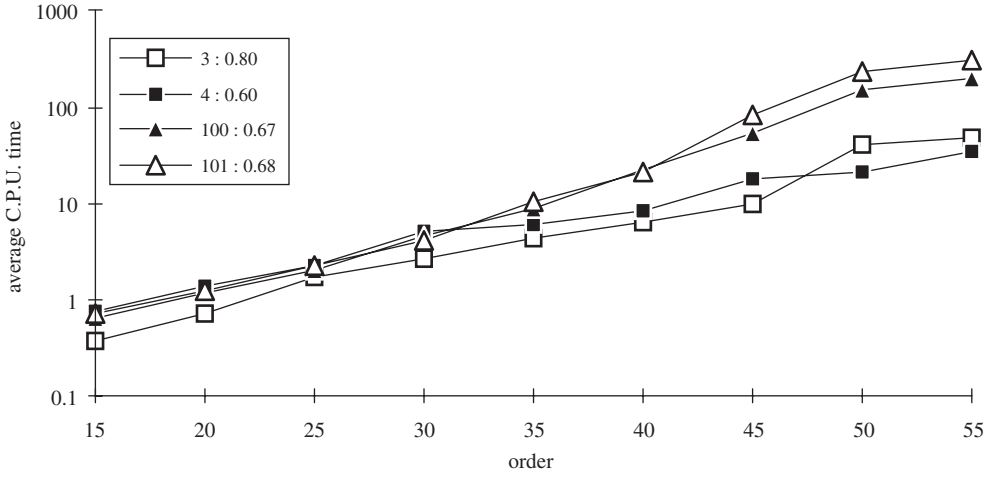

Fig. 8. Time to solve problems got by the aggregation of linear orders.

For the first three kinds of tournaments, the aggregated relations are only supposed to be complete and asymmetric (for $i \leqslant j$, x_i is preferred to x_j if and only if x_j is not preferred to x_i). Moreover, we assume that, for each pair of vertices $\{xi, x_j\}$, with $1 \leqslant i < j \leqslant n$, a judge decides that he or she prefers x_i to x_j with a probability equal to

$$\pi_1 + \frac{(\pi_2 - \pi_1)(j - i)}{n - 1},$$

where π_1 and π_2 are parameters satisfying $0.5 \leqslant \pi_1 \leqslant \pi_2 \leqslant 1$; of course, if he or she does not prefer x_i to x_j, he or she prefers x_j to x_i. Notice that, when $\pi_1 = \pi_2$, the orientation of the arc between x_i and x_j does not depend on i and j. If moreover $\pi_1 = \pi_2 = 0.5$, we get a tournament in which the arc between x_i and x_j has the same probability to be (x_i, x_j) or (x_j, x_i), what leads usually to difficult tournaments, with low transitivity indices. For example, Slater32 is such a tournament (associated to only one judge), with a transitivity index equal to 0.0875. When we deal with several judges, the weight $w_{i,j}$ is, as before

$$w_{i,j} = \text{number of judges preferring } x_i \text{ to } x_j - \text{number of judges preferring } x_j \text{ to } x_i.$$

We denote π_1–π_2 *problem* a problem corresponding with the above description, with the parameters π_1 and π_2.

Still for the first three kinds of problems, the number of judges is successively chosen as 1, 3, 15 and 25 (thus, when the number of judges is 1, we get an unweighted tournament, that is an instance of Slater's problem; it is the case of Slater32). For instance, in the previous section, the tournament Judges100 is a 0.5–0.85 problem with 50 judges. The experiments were performed for several orders, increasing from 15 to a maximum value depending on the time needed to solve these problems. In the legend, we give the number of judges corresponding with a curve and the average value of the transitivity index: a first average is computed over the 30 tournaments with the same order, then a second average is done over the tested orders; it is this last value which is given; in fact, the transitivity index decreases (in average) when the order increases; in our experiments, the relative decrease from the order 15 to the maximum studied order is of some percents.

We can observe that we get usually more difficult tournaments with $(\pi_1, \pi_2) = (0.5, 1)$ than with $(0.75, 0.75)$ and still more difficult ones with $(0.5, 0.5)$: we can notice it through the values of the transitivity indices as well as the values of the orders n that we can deal with. For given values of π_1 and π_2, the π_1–π_2 problems are always more difficult for only one judge (Slater's problem), and are easier as the number of judges increases, especially for 0.5–1 and 0.75–0.75 problems; for the 0.5–0.5 problems, we explain this by the fact that the weights become more different when the number of judges increases; for the 0.5–1 problems and the 0.75–0.75 problems, it is clear that, when the number of judges increases, the tournament becomes more and more "transitive", getting closer to the order $x_1 > x_2 > \cdots > x_n$; the transitivity index confirms this growing transitivity.

Problems of the last kind come from the aggregation of linear orders. Each judge chooses an order on the vertices randomly, with a uniform probability, and these orders are aggregated as we did for the previous problems when computing $w_{i,j}$ as the number of judges who prefer x_i to x_j minus the number of judges who prefer x_j to x_i. In the legend of Fig. 8, the information is the same as for the figures above. From these results, we see that there is no significant difference with respect to the parity of the number of judges and that the problems are a bit more difficult with 100 or 101 judges than with 3 or 4 judges, what is not surprising (the problem is even trivially polynomial if the number of judges is 1 or 2; it is why we did not try these values). However, the transitivity indices remain rather high, always more than 0.5, what allows us to solve problems with up to 55 vertices (and sometimes more) within a reasonable CPU time.

From these results, it appears that the maximum value of the number n of vertices of the tournaments T of which we can find a median order (or a Slater order) in a "reasonable" CPU time (let us say about 1 CPU hour) depends on the type of T. Broadly speaking, for "very difficult" tournaments T with $\tau(T)$ about 0.1 like the ones of 0.5–0.5 problems for which the two orientations of each arc have the same probability, this maximum value ranges from more than 30 (for unweighted tournaments, that is for Slater's problem) to about 40 (sometimes more); for easier problems like 0.75–0.75 ones, it goes beyond 100 (except for Slater's problem, for which the maximum value is about 45 in our experiments). These ranges compare favorably with the ones recalled at the beginning of Section 4.

A last remark is about the noising method, which is applied before the branch-and-bound method, using always the same self-tuning of the parameters (see [22]). Over the 5790 tournaments tested to draw the above curves and with up to 100 vertices, the noising method found the optimal solution in all the cases but six (one tournament corresponding with a 0.5–1 problem, and five with problems got by the aggregation of linear orders; notice that a second application of the noising method succeeded in solving these 6 tournaments exactly), and thus it gave the optimal solution for about 99.9% of the studied problems. Furthermore, for our problems, the noising method never takes more than a few seconds. As it is a stochastic method, if we try it twice, it is almost sure to get an exact solution, at least for the orders considered here.

7. Conclusion

Data coming from life problems often give tournaments with a transitivity index very close to 1, and so, it is often possible to solve these real-life problems with orders rather high by means of our software. Nevertheless, when the number of vertices is more than 100, it can become difficult to solve the problem exactly, because the necessary memory space (mainly for the Lagrangean multipliers) or the required CPU time can become too large. But in this case, the noising method usually gives a good solution, often exact; therefore, what becomes difficult is to check that the solution provided by the noising method is optimal.

In our method, the parameters of the Lagrangean relaxation and of the noising method are automatically computed from only the transitivity index, the number of vertices and the weights of the arcs of the tournament, so that the user

has nothing to tune. For a given tournament, the choice of these parameters may not be the best: some other values could increase the chance to have the optimal solution with the noising method, or could decrease the CPU time needed to perform the branch-and-bound method. But, tuning the best parameters consumes time (and, from a practical point of view, it is in fact impossible to find the best parameters). Moreover, such a tuning is very difficult to do for somebody who is not very familiar with these techniques. So, we think that it is better to have software depending on no parameter (except of course the data themselves).

We can say that "union makes strength". With a good bound found at the beginning by the noising method and with many components to cut nodes in the branching method, it is possible to solve rather big and complex problems. The help of Lagrangean relaxation seems to be crucial except for very small or easy problems, but it takes a long time, and it is necessary, in order to save time, to use quicker ways to cut nodes when it is possible. It is what we tried to do in our software. This one is freely available at the following address: http://www.enst.fr/∼charon/tournament/median.html. We would be very happy if other researchers would use it to solve their problems. Of course, any comment about it would be very welcome.

References

[1] E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, New York, 1997.
[2] D. Arditti, Un nouvel algorithme de recherche d'un ordre induit par des comparaisons par paires, in: E. Diday et al. (Eds.), Data Analysis and Informatics III, North Holland, Amsterdam, 1984, pp. 323–343.
[3] J.-P. Barthélemy, Social welfare and aggregation procedures: combinatorial and algorithmic aspects, in: F. Roberts (Ed.), Graph Theory and Combinatorics in the Biological and Social Sciences, IMA Series, Springer, Berlin, 1989.
[4] J.-P. Barthélemy, C. Flament, B. Monjardet, Ordered sets and social sciences, in: I. Rival (Ed.), Ordered Sets, D. Reidel, Dordrecht, 1989, pp. 721–758.
[5] J.-P. Barthélemy, A. Guénoche, O. Hudry, Median linear orders: heuristics and a branch and bound algorithm, European J. Oper. Res. 41 (1989) 313–325.
[6] J.-P. Barthélemy, B. Leclerc, B. Monjardet, On the use of ordered sets in problems of comparison and consensus of classifications, J. Classification 3 (1986) 187–224.
[7] J.-P. Barthélemy, B. Monjardet, The median procedure in cluster analysis and social choice theory, Math. Social Sci. 1 (1981) 235–267.
[8] J.-P. Barthélemy, B. Monjardet, The median procedure in data analysis: new results and open problems, in: H.H. Bock (Ed.), Classification and Related Methods of Data Analysis, North Holland, Amsterdam, 1988.
[9] J.E. Beasley, Lagrangean relaxation, in: C.R. Reeves (Ed.), Modern Heuristic Techniques for Combinatorial Problems, McGraw Hill, London, 1995, pp. 243–303.
[10] J.-C. Bermond, Ordres à distance minimum d'un tournoi et graphes partiels sans circuits maximaux, Math. Sci. Humaines 37 (1972) 5–25.
[11] J.-C. Bermond, Y. Kodratoff, Une heuristique pour le calcul de l'indice de transitivité d'un tournoi, RAIRO 10 (3) (1976) 83–92.
[12] K. Boenchendorf, Reihenfolgenprobleme/Mean-flow-time sequencing, Mathematical Systems in Economics, vol. 74, Verlagsgruppe Athanäum, hain, Scriptor, Hanstein, 1982.
[13] V.N. Burkov, V.O. Groppen, Branch cuts in strongly connected graphs and permutation potentials, Automat. Remote Control 6 (1972) 111–119.
[14] J.S. de Cani, Maximum likelihood paired comparison ranking by linear programming, Biometrika 3 (1969) 537–545.
[15] J.S. de Cani, A branch and bound algorithm for maximum likelihood paired comparison ranking, Biometrika 59 (1972) 131–135.
[16] I. Charon, A. Germa, O. Hudry, Encadrement de l'indice de Slater d'un tournoi à l'aide de ses scores, Math. Inform. Sci. Humaines 118 (1992) 53–68.
[17] I. Charon, A. Guénoche, O. Hudry, F. Woirgard, A bonsaï branch and bound method applied to voting theory, in: Proceedings of the International Conference on Ordinal and Symbolic Data Analysis (OSDA), Collection Studies in Classification, Data Analysis, and Knowledge Organization, Springer, Berlin, 1996, pp. 309–318.
[18] I. Charon, A. Guénoche, O. Hudry, F. Woirgard, New results on the computation of median orders, Discrete Math. 165–166 (1997) 139–153.
[19] I. Charon, O. Hudry, The noising method: a new method for combinatorial optimization, Oper. Res. Lett. 14 (1993) 133–137.
[20] I. Charon, O. Hudry, Lamarckian genetic algorithms applied to the aggregation of preferences, Ann. Oper. Res. 80 (1998) 281–297.
[21] I. Charon, O. Hudry, The noising methods: a generalization of some metaheuristics, European J. Oper. Res. 135 (1) (2001) 86–101.
[22] I. Charon, O. Hudry, Self-tuning of the noising methods, submitted for publication.
[23] I. Charon, O. Hudry, F. Woirgard, Ordres médians et ordres de Slater des tournois, Math. Inform. Sci. Humaines 133 (1996) 23–56.
[24] T. Christof, G. Reinelt, Combinatorial Optimization and Small Polytopes, Top 4 (1) (1996) 1–64.
[25] W.D. Cook, A.L. Saipe, Committee approach to priority planning: the median ranking method, Cahiers Centre Détudes Rech. Opér. 18 (3) (1976) 337–352.
[26] M. Dell'Amico, F. Maffioli, S. Martello (Eds.), Annotated Bibliographies in Combinatorial Optimization, Wiley, Chichester, 1997.
[27] J.A. Flueck, J.F. Korsh, A branch search algorithm for maximum likelihood paired comparison ranking, Biometrika 61 (3) (1974) 621–626.
[28] M. Grötschel, M. Jünger, G. Reinelt, A cutting plane algorithm for the linear ordering problem, Oper. Res. 32 (1984) 1195–1220.
[29] M. Grötschel, M. Jünger, G. Reinelt, Optimal triangulation of large real-world input-output-matrices, Statist. Hefte 25 (1984) 261–295.
[30] M. Grötschel, M. Jünger, G. Reinelt, Acyclic subdigraphs and linear orderings: polytopes, facets, and a cutting plane algorithm, in: I. Rival (Ed.), Graphs and Orders, D. Reidel Publishing Company, Dordrecht, 1985, pp. 217–264.

[31] A. Guénoche, Un algorithme pour pallier l'effet Condorcet, RAIRO 11 (1) (1977) 77–83.
[32] A. Guénoche, A. B. C. D. : logiciel d'analyses booléennes et combinatoires des données, notice d'utilisation, G. R. T. C., Marseille, 1986.
[33] A. Guénoche, Order at minimum distance of a valued tournament, communication to Modélisation, Analyse et Agrégation des Préférences et des Choix TRAP 3, Marseille-Luminy, 1988.
[34] A. Guénoche, How to choose according to partial evaluations, in: B. Bouchon-Meunier et al. (Ed.), Advances in Intelligent Computing, IPMU'94, Lecture Notes in Computer Sciences, vol. 945, Springer, Berlin, 1995, pp. 611–618.
[35] A. Guénoche, Vainqueurs de Kemeny et tournois difficiles, Mathématiques, Informatique et Sciences Humaines 133 (1996) 57–66.
[36] A. Guénoche, B. Vandeputte-Riboud, J.-B. Denis, Selecting varieties using a series of trials and a combinatorial ordering method, Agronomie 14 (1994) 363–375.
[37] L. Hubert, Seriation using asymmetric proximity measures, British J. Math. Statist. Psychol. 29 (1976) 32–52.
[38] O. Hudry, Recherche d'ordres médians : complexité, algorithmique et problèmes combinatoires, Ph.D. thesis, ENST, Paris, 1989.
[39] O. Hudry, Tournois et optimisation combinatoire, mémoire d'Habilitation à diriger des recherches, ENST, Paris, 1998.
[40] M. Jünger, Polyhedral combinatorics and the acyclic subdigraph problem, Heldermann Verlag, Berlin, 1985.
[41] R. Kaas, A branch and bound algorithm for the acyclic subgraph problem, European J. Oper. Res. 8 (1981) 355–362.
[42] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[43] J.G. Kemeny, Mathematics without numbers, Daedalus 88 (1959) 577–591.
[44] M.G. Kendall, B. Babington Smith, On the method of paired comparisons, Biometrika 33 (1940) 239–251.
[45] B. Korte, W. Oberhofer, Zwei Algorithmen zur Lösung eines komplexen Reihenfolgeproblems, Unternehmensforschung 12 (1968) 217–231.
[46] B. Korte, W. Oberhofer, Zur Triangulation von Input-Output-Matrizen, Jahrbuch. Nationaloekon. Statist. 182 (1969) 398–433.
[47] G. Laporte, I.H. Osman, Metaheuristics: a bibliography, Ann. Oper. Res. 63 (1996) 513–623.
[48] H.W. Lenstra, Jr., The acyclic subgraph problem, Technical Report BW26, Mathematisch Centrum, Amsterdam, 1973.
[49] J.-F. Marcotorchino, P. Michaud, Optimisation en analyse ordinale de données, Masson, Paris, 1979.
[50] M. Minoux, Mathematical Programming: Theory and Algorithms, Wiley, New York, 1986.
[51] J.W. Moon, Topics on tournaments, Holt, Rinehart and Winston, New York, 1968.
[52] J.P.N. Phillips, A procedure for determining Slater's *i* and all nearest adjoining orders, British J. Math. Statist. Psychol. 20 (1967) 217–225.
[53] J.P.N. Phillips, A further procedure for determining Slater's *i* and all nearest adjoining orders, British J. Math. Statist. Psychol. 22 (1969) 97–101.
[54] C. Reeves (Ed.), Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, London, 1993.
[55] G. Reinelt, The linear ordering problem: algorithms and applications, Research and exposition in mathematics, vol. 8, Heldermann, Verlag, Berlin, 1985.
[56] R. Remage, W.A. Thompson, Maximum likelihood paired comparison rankings, Biometrika 53 (1966) 143–149.
[57] P. Slater, Inconsistencies in a schedule of paired comparisons, Biometrika 48 (1961) 303–312.
[58] A.W. Tucker, On directed graphs and integer programs, 1960 Symposium on Combinatorial Problems, Princeton University, USA, 1960.
[59] U. Tüshaus, Aggregation binärer Relationen in der qualitativen Datenanalyse, Mathematical Systems in Economics, vol. 82, Verlagsgruppe Athenäum, Hain, Hanstein, 1983.
[60] H. Wessels, Triangulation und Blocktriangulation von Input-Output Tabellen, Deutsches Institut für Wirtschaftsforschung: Beiträge zur Strukturforshung, Heft, vol. 63, Berlin, 1981.
[61] F. Woirgard, Recherche et dénombrement des ordres médians des tournois, Ph.D. Thesis, ENST, Paris, 1997.
[62] D.H. Younger, Minimum feedback arc sets for a directed graph, IEEE Trans. Profes. Tech. Group Circuit Theory 10 (2) (1963) 238–245.