# Applications of modern heuristic search methods to pattern sequencing problems*

Andreas Fink† Stefan Voß†

May 1998

**Scope and Purpose**—Pattern sequencing problems have important applications, especially in the field of production planning. Those problems generally consist of finding a permutation of predetermined production patterns (groupings of some elementary order types) with respect to different objectives. These objectives may represent, e.g., handling costs or stock capacity restrictions, which usually leads to $\mathcal{NP}$-hard problems. Thus, the use of heuristics to construct respective pattern sequences is generally assumed to be appropriate.

**Abstract**—This article describes applications of modern heuristic search methods to pattern sequencing problems, i.e., problems seeking for a permutation of the rows of a given matrix with respect to some given objective function. We consider two different objectives: Minimization of the number of simultaneously open stacks and minimization of the average order spread. Both objectives require the adaptive evaluation of changed solutions to allow an efficient application of neighbourhood search techniques. We discuss the application of several modern heuristic search methods and present computational results.

## 1 Introduction

Pattern sequencing problems seek for an optimal permutation of the rows of a given matrix with respect to some given objective function. More formally, pattern sequencing problems may be stated as follows: Let $C$ be an $m \times n$ matrix with integer elements $c_{ij}$, $1 \leq i \leq m$, $1 \leq j \leq n$. The objective is to construct a permutation $\Pi = (\pi_1, \ldots, \pi_m)$ of the rows (=patterns) of this matrix that minimizes some given objective function. Here $\pi_i$ denotes the pattern that is positioned at the $i$-th position. Thus the new $i$-th row of the matrix is now $(c_{\pi_i,1}, \ldots, c_{\pi_i,n})$. Objective functions

*to appear in *Computers & Operations Research* ©

†Technische Universität Braunschweig; Allg. BWL, Wirtschaftsinformatik und Informationsmanagement; Abt-Jerusalem-Str. 7, D-38106 Braunschweig, Germany; email: {a.fink, stefan.voss}@tu-bs.de.

considered here differ from travelling salesman like problems by the non-locality of the evaluation; i.e., the evaluation of a permutation can not be computed by using values that only depend on patterns adjacent to each other. The interpretations of these problems are generally based on the assumption that the columns represent some elementary "orders" and that the patterns represent predetermined groupings of these orders. In the sequel we first present several relevant objective functions to be minimized, each with an example representative of real life applications, before giving theoretical results and discussing the rest of the paper.

## 1.1 Applications

Pattern sequencing problems have important applications, especially in the field of production planning. In the following we summarize some of these problem types.

### 1.1.1 Simultaneously open stacks (PSP-SOS)

Consider the production of the order patterns given by $C$. An entry $c_{ij}$ of $C$ represents the frequency of order type $j$ in pattern $i$; in the following it is only relevant whether $c_{ij} > 0$. An order $j$ is called open, if its production has been started but not yet been finished. More formally, an order $j$ may be defined open at position $i'$ of the pattern sequence if $(\sum_{i=1}^{i'} c_{\pi_i,j})(\sum_{i=i'}^{m} c_{\pi_i,j}) > 0$. The PSP-SOS consists of constructing a sequence of the order patterns with respect to minimizing the number of simultaneously open order stacks. This objective may refer to corresponding handling costs or restrictions like, e.g., a limited number of containers to be served simultaneously (provided that each order is assigned to one container) or restrictions concerning stock capacity. Yanasse [1] gives an example in the field of wood cutting.

### 1.1.2 Average order spread (PSP-AOS)

Starting from the same assumptions as for the PSP-SOS with an additional frequency vector $\alpha$ with $\alpha_i$ representing the frequency of pattern $i$, the objective is now to find a sequence of order patterns with respect to the following policy: as there may be some handling costs connected with each open order, we may strive for a minimization of the average length each order is open, the so-called average order spread. Formally, the average order spread of a permutation $\Pi$ is defined as

$$AOS(\Pi) = \frac{1}{n} \sum_{j=1}^{n} \left( \sum_{i=\min\{k|c_{\pi_k,j}>0\}}^{\max\{k|c_{\pi_k,j}>0\}} \alpha_{\pi_i} - 1 \right).$$

Madsen [2] gives an example from the field of cutting stock problems, where the patterns are results of a prior optimization step with the aim to minimize waste

of material. A possible modification would be to consider the minimization of the maximum order spread.

### 1.1.3 Actor costs (AC) of shooting schedules

Consider the shooting of a movie (cf. [3, 4]). The patterns of $C$ define the scenes to be done. The binary matrix elements are interpreted as follows: $c_{ij}$ equals 1 if and only if actor $j$ is needed in scene $i$. A cost vector $\gamma$ represents the costs $\gamma_j$ of actor $j$ per time unit. An actor is paid the time from the first to the last scene where he or she is needed. A frequency vector $\alpha$ may define the length $\alpha_i$ (in time units) the corresponding scene $i$ requires to be completed. The objective is to find a sequence of scenes that minimizes the total actor costs. The actor costs (AC) of a permutation $\Pi$ are defined as

$$AC(\Pi) = \sum_{j=1}^{n} \gamma_j \sum_{i=\min\{k|c_{\pi_k,j}>0\}}^{\max\{k|c_{\pi_k,j}>0\}} \alpha_{\pi_i} \ .$$

The objective $AC(\Pi)$ may be referred to as a generalization of the objective $AOS(\Pi)$. More specifically, it may be seen as a weighted version of $AOS(\Pi)$ with $\gamma_j = 1$ for all $j = 1, \ldots, n$.

### 1.1.4 Further problem types

There are objective functions for sequencing problems that lead to the classical travelling salesman problem, e.g., minimize the discontinuities of the production of the order types; there the "distance" between two patterns $i$ and $k$ is defined as the number of order types that are contained in exactly one of these patterns.

Sequencing problems related to those described above also exist in various other domains, e.g., sequencing of DNA segments [5], ordering of some "information" items [6, 7, 8], or data base organization [9].

### 1.1.5 Example

Consider the following example with $m = 4$ patterns and $n = 4$ order types, where these patterns are produced according to the given sequence $(1, 2, 3, 4)$:

$$C = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

While producing the third pattern all four order types are simultaneously open. Assuming that all pattern frequencies are equal to one, the average order spread of the given sequence is $(0 + 2 + 3 + 3)/4 = 2$. Further on, we will use this example to illustrate potential improvements due to advanced heuristics.

## 1.2 Theoretical results

In this section we discuss the computational complexity of the problem types defined above. As PSP-AOS is assumed to be $\mathcal{NP}$-hard in some papers without giving a proof or a literature reference (cf., e.g., [10]) while the problem complexity is not even touched in other papers (cf., e.g., [11]), we give a simple proof of the $\mathcal{NP}$-hardness of a special case of PSP-AOS, which also shows that PSP-AC is $\mathcal{NP}$-hard. The proof is based on the idea of [3] as cited in [4].

*Proposition:* PSP-AOS is $\mathcal{NP}$-hard.

*Proof:* The decision version of PSP-AOS is in $\mathcal{NP}$. We transform the *Optimal Linear Arrangement Problem* to the decision version of PSP-AOS. The optimal linear arrangement problem may be stated as follows: Given is a graph $G = (V, E)$ and a positive integer $K$. Is there a one-to-one function $f : V \to \{1, 2, \ldots, |V|\}$ such that $\Psi(f) = \sum_{(u,v)\in E} |f(u) - f(v)| \leq K$? The optimal linear arrangement problem is $\mathcal{NP}$-complete, even if $G$ is bipartite (cf. [12]).

The transformation from the optimal linear arrangement problem to the decision version of PSP-AOS is as follows. Let $V = \{1, \ldots, m\}$ and $E = \{e_1, \ldots, e_n\}$. The rows (columns) of the matrix $C_{m\times n}$ correspond to the vertices (edges) of $G$. The binary entries of $C$ are defined as follows: $c_{ij}$ equals 1 if and only if edge $e_j$ is incident to vertex $i$. The pattern frequencies $\alpha_i$ are all set to one. To prove that this is indeed a transformation, we show that a function $f$ with the specified property exists if and only if there is a permutation $\Pi$ of the rows of $C$ with $AOS(\Pi) \leq K/n$.

The function $f$ constitutes a permutation $\Pi$ of the rows of the matrix. The summands $|f(u)-f(v)|$ of $\Psi(f)$ represent the "distance" between the two non-zero entries of the corresponding columns of $C$. So we can state the following equivalence:

$$\begin{aligned}
\Psi(f,\Pi) &= \sum_{(u,v)\in E} |f(u) - f(v)| \\
&= \sum_{j=1}^{n} \left(\max\{k|c_{\pi_k,j} > 0\} - \min\{k|c_{\pi_k,j} > 0\}\right) \\
&= \sum_{j=1}^{n} \left( \sum_{i=\min\{k|c_{\pi_k,j}>0\}}^{\max\{k|c_{\pi_k,j}>0\}} \alpha_{\pi_i} - 1 \right) \\
&= n\, AOS(\Pi)\ .
\end{aligned}$$

As we have constructed a polynomial tranformation from the optimal linear arrangement problem to PSP-AOS, the decision version of PSP-AOS is $\mathcal{NP}$-complete, i.e., PSP-AOS is $\mathcal{NP}$-hard. □

The transformation provides us with the $\mathcal{NP}$-hardness for a special type of PSP-AOS with pattern frequencies of one and each order type produced in exactly two patterns. Further, due to the bipartite graph $G$ the patterns may be classified in two sets, with each order type produced in patterns from both sets.

Whether PSP-SOS is $\mathcal{NP}$-hard or efficiently solvable is open (cf. [1]).

## 1.3 Earlier work

Yanasse [1] surveys some open problems concerning the PSP with the goal to minimize the number of simultaneously open stacks (PSP-SOS). The question, whether PSP-SOS is $\mathcal{NP}$-hard or efficiently solvable is open, whereas general pattern sequencing problems are $\mathcal{NP}$-hard. He gives some conjectures concerning the relationships of PSP-SOS with similar problems. Furthermore, he describes lower bounds and a Branch-and-Bound scheme for PSP-SOS. However, he does not present any computational results.

Yuen [13, 11] compares different priority rules to construct solutions for PSP-SOS. Yuen and Richardson [14] present methods to establish the optimality of sequences for PSP-SOS (lower bounds, disjoint subgraphs, exhaustive search).

In a case study presenting problems from the glass industry, Madsen [2] applies travelling salesman routines to solve PSP-AOS like problems.

Foerster [15] compares priority rules and different metaheuristics for PSP-AOS. Foerster and Wäscher [10] examine especially the application of 3-opt and simulated annealing for PSP-AOS.

Chen et al. [3] pose the problem to minimize the actor costs of shooting schedules (PSP-AC). Nordström and Tufekci [4] present a genetic algorithm with computational results for PSP-AC.

## 1.4 Motivation and outline

A motivation for this paper is to deal with the different problem types of the research field described above in a unifying way. Here we consider the two primary representatives of the most relevant objectives: minimization of the average order spread (PSP-AOS) and minimization of the number of simultaneously open stacks (PSP-SOS). The use of heuristics is generally assumed to be appropriate, as these objective functions usually constitute some imprecise, derived goals. For example, the objective of minimizing the average order spread (PSP-AOS) may approximately substitute the goal "minimization of handling costs", which may generally not be exactly defined and quantified. Besides, the problem types considered here belong to the class of $\mathcal{NP}$-hard problems (PSP-AOS) or there is no known efficient algorithm (PSP-SOS).

As described in Section 1.3, the research done so far mostly lacks the application of advanced local search based metaheuristics. Only Foerster [15] applies corresponding methods, however, concerning tabu search, with modest success. This is partly due to the difficulties to evaluate the solution space within a neighbourhood search in an efficient way. In the next section we first examine the problem of efficiently constructing good starting solutions; then we present efficient algorithms to evaluate

neighbours. In Section 3 we discuss the heuristics applied; we focus on different tabu search heuristics. Afterwards, we present computational results and lastly draw some conclusions.

# 2 Solution space

## 2.1 Construction of starting solutions

The application of local search heuristics requires the construction of initial feasible solutions. This can be accomplished by, e.g., starting with the identity permutation, a random permutation, or some heuristically determined starting solution. Which of these strategies is useful generally has to be decided by experimental computations (cf. Section 4). Here we consider starting with the identity permutation or with a permutation determined by two different construction heuristics: cheapest insertion and cheapest insertion of worst pattern, where the latter may be referred to as farthest insertion, too.

The cheapest insertion heuristic works as follows. Start with a partial sequence $\Pi =< i >$, that includes only one pattern $i$ (here pattern 1). Now build successively a complete permutation by choosing in each iteration $k = 2, \ldots, m$ the best combination under consideration of the remaining $m - k + 1$ patterns and all $k$ insertion positions. That is, we have to evaluate the objective function increase for all these combinations at each iteration. Generally this may be accomplished in $O(m^3)$ time only if it is possible to evaluate each insertion position in constant time; so the exact time complexity depends on the evaluation function. A trivial evaluation for the objective functions considerd here leads to $O(m^4 n)$ time. Due to these seemingly prohibitive costs the cheapest insertion heuristic was neglected in [15].

However, we may reduce the average time needed considerably by performing calculations as shown for PSP-AOS in Algorithm 2 in the Appendix. There we use vectors *first* and *last* that represent the current first and last occurrences, respectively, of each order in the pattern sequence. $\Delta$ is computed as the increase of the order spread when inserting pattern $i$ before position $p$. For each order type $j$ we calculate the increase of the order spread by distinguishing four cases: i) First production of order $j$. ii) New earliest production of order $j$. iii) Order $j$ to be produced neither earliest nor latest. iv) New latest production of order $j$.

The respective evaluation for PSP-SOS is shown in Algorithm 3 in the Appendix. There we further use a vector *open* that holds for each position of the partial sequence the number of simultaneously open orders. The number of simultaneously open orders for the inserted pattern is computed as $adaptedOpen[0]$. However, the evaluation of an insertion as presented in Algorithm 3 by comparing the maximal values of both vectors may be too rough to provide appropriate information about the quality of potential

insertions. In Section 2.4 we present a modified evaluation function for neighbourhood solutions, that might also be used here.

Apart from these cheapest insertion heuristics we also used a modified approach, where in each step that pattern is selected for insertion whose cheapest insertion leads to a maximal increase of the objective function. This algorithm may be termed as *cheapest insertion of worst pattern*. This approach is similar to the *farthest insertion* method for the travelling salesman problem. The motivation for the application of such strategies is due to the idea of performing the important decisions (those that lead to a high increase of the objective function value) first.

## 2.2 Neighbourhood

Consider a permutation $\Pi$ as a chain of patterns, each connected by an edge, with two dummy patterns that represent a virtual fixed first (0) and last ($m$+1) pattern. Now a 2-exchange move is defined for a pair $(p_1, p_2)$ with $0 \leq p_1$ and $p_1 + 2 \leq p_2 \leq m$ as the deletion of the edges $(\pi_{p_1}, \pi_{p_1+1})$ and $(\pi_{p_2}, \pi_{p_2+1})$ and the inclusion of the edges $(\pi_{p_1}, \pi_{p_2})$ and $(\pi_{p_1+1}, \pi_{p_2+1})$. The number of neighbours of a given solution for this neighbourhood may be computed as

$$\sum_{i=1}^{m-1} i = \frac{m(m-1)}{2} = O(m^2).$$

Here we concentrate on the 2-exchange neighbourhood, as further useful neighbourhoods, that may be determined by other possible move definitions (e.g. 3-exchange), might lead to a great increase of running time of the advanced heuristics considered below.

As the neighbourhood considered here has size $O(m^2)$, it is crucial to perform the evaluation of a neighbourhood in an adaptive way to reach sufficiently efficient implementations. Though the worst case of the algorithms presented below still leads to $O(m^3 n)$ time for the evaluation of a neighbour, the average time is significantly reduced.

## 2.3 Neighbourhood evaluation for PSP-AOS

In order to achieve efficiency we define the vectors *first* and *last*, which represent the first and last occurrences, respectively, of each order in the current pattern sequence. These vectors have to be adapted each time a move is performed; this may be done in connection with the reordering of the patterns with no significant cost overhead.

To evaluate a neighbour $(p_1, p_2)$ of a given solution we have to distinguish for all orders $j$ the cases shown in Figure 1. Neighbours of type a, c, d or f do not lead to a different objective function value; only for neighbours of type b or e does the order spread change. The evaluation of a neighbour may be done as shown in Algorithm 4 in

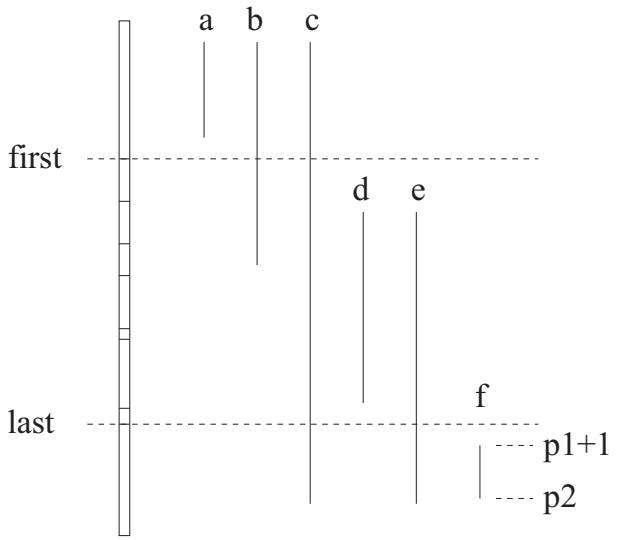


Figure 1: Cases to differentiate when evaluating neighbours.

the Appendix. Each time when *first* or *last* are affected, the respective order spread changes. That is, for case b we have to subtract the frequencies of those patterns that would be positioned before the new first position, and to add the frequencies of those patterns that are currently positioned before the old first position but would now be included in the order spread. In case e we have to subtract the frequencies of those patterns that would be positioned after the new last position, and to add the frequencies of those patterns that are currently positioned after the old last position but would now be included in the order spread.

## 2.4 Neighbourhood evaluation for PSP-SOS

Besides the vectors *first* and *last* we further introduce a vector *open*, that stores for each position $i$ the number of simultaneously open orders for the current solution. To evaluate a neighbour $(p_1, p_2)$ of a given solution, we have to distinguish for all orders $j$ the cases shown in Figure 1. Neighbours of type a, d and f do not change the status of an order for any position $i$. Only neighbours of type b, c or e may change *open*, and so the evaluation (the maximum element of *open*). The efficient evaluation of a neighbour may be accomplished as shown in Algorithm 5 in the Appendix. We briefly describe the corresponding calculation for case b: First, we determine the new earliest production of order $j$. Dependent on whether this leads to a move of the production of order $j$ up ahead, we must increase the *open* vector accordingly, or otherwise decrease it.

As already noted for construction heuristics in Section 2.1, the simple evaluation of a neighbour by the potential change of the objective function value does not provide enough information to guide local search methods into promising search regions. For instance, computational experiments have shown that steepest descent often immediately terminates with the starting solution, as there is no move that leads to an improvement of the objective function value, i.e., a direct decrease of the maximum value of the *open* vector. However, there is often a sequence of moves that successively decrease some entries of the *open* vector till finally a reduction of the maximum value becomes possible. So we adopt the following evaluation of a neighbour, that is based on the idea to "smooth" the solution space by evaluating not only the maximum value but the most significant difference between two solutions. Assume both vectors *open* and *adaptedOpen* as sorted descending. If *open* equals *adaptedOpen* the evaluation is zero. Otherwise define $\eta$ as the first position with different entries, i.e. $\eta = \min\{i | open[i] \neq adaptedOpen[i], i = 1, \ldots, m\}$. With $\delta = open[\eta] - adaptedOpen[\eta]$ the evaluation is computed as

$$\begin{aligned} &\frac{1}{\eta+1} + \frac{\delta}{m(\eta+2)(\eta+3)} && \text{for} \quad \delta > 0 \ , \\ &\frac{-1}{\eta+1} + \frac{\delta}{m(\eta+2)(\eta+3)} && \text{for} \quad \delta < 0 \ . \end{aligned}$$

That is, the position of the most significant change of the *open* vector dominates the evaluation of a neighbour. If $\eta$ is equal for two neighbours the second term includes the change of that value as secondary criterion.

# 3 Improvement methods

Improvement procedures for pattern sequencing problems may be based on local search considering those neighbourhood ideas described in the previous section. Steepest descent selects in each iteration the best neighbour until no improving move is possible. For the 2-exchange neighbourhood this leads to a so-called 2-opt locally optimal solution. As steepest descent may lead to local optima of non-satisfying solution quality, in the sequel we consider the application of modern heuristic search methods, i.e., simulated annealing and various tabu search approaches.

Consider the simple example from Section 1.1.5 regarded as PSP-AOS. Starting from the given sequence $(1, 2, 3, 4)$ the best 2-exchange move leads to the new sequence $(2, 1, 3, 4)$ with an average order spread of 1.5, which represents a local optimum. Thus, a simple steepest descent approach will get stuck in this solution. However, the global optimum $(3, 1, 4, 2)$ (or the reverse sequence) has an average order spread of 1.25. All methods discussed below easily obtain this solution.

## 3.1 Simulated annealing

Simulated annealing is a local search based metaheuristic that randomly allows deteriorating moves to overcome local optimality (cf. [16]). In every iteration a potential move is randomly selected; this move is accepted if it leads to a solution with a better objective function value than the current solution, otherwise the move is accepted with a probability that depends on the deterioration $\delta$ of the objective function value. The probability of acceptance is computed here according to the Boltzmann function as $e^{-\delta/T}$ using a temperature $T$. The temperature is reduced through multiplication by a parameter $a$ according to a cooling schedule. This reduction is performed after a repetition interval that is increased by multiplication with a parameter $b$ after every reduction. Here we follow the procedure and use the parameters described in [10]: the initial temperature is set to half of the objective function value of the starting solution, the repetition interval is initialized by $m$, $a$ by 1.15 and $b$ by 0.75.

## 3.2 Tabu search

The basic idea of tabu search (cf. [17]) is to use information about the search history to guide local search approaches to overcome local optimality. Primarily, this is done by either statically or dynamically prohibiting certain moves; the different tabu search strategies differ especially in the way how the tabu criteria are defined. A general description of a tabu search frame may be presented as shown in Algorithm 1 for a given starting solution $s$ and a tabu criterion that is represented by the object TabuMemory. A neighbour (move) is called *admissible*, if it is not tabu or if an aspiration criterion is fulfilled (e.g., the move leads to a neighbour which represents a better solution than found so far). Eventually, a tabu search method may incorporate diversifying moves that should drive the search into new regions when it might be trapped in a certain area of the solution space. In the following we shortly describe different tabu criteria that differ especially in the way they use the information provided (performed moves, traversed solutions) to establish the tabu status.

### 3.2.1 Static tabu search

The basic idea of static tabu search is to prohibit the inversion of performed moves for a fixed number of iterations (tabu duration). The possible disadvantage of such a heuristic is that some parameters (here the tabu duration) have to be set in advance.

Considering a performed move $(p_1, p_2)$ we store the attributes that represent the inserted edges, i.e. $(\pi_{p_1}, \pi_{p_2})$ and $(\pi_{p_1+1}, \pi_{p_2+1})$, in a tabu list of fixed length. To obtain the current tabu status of a neighbour that is represented by $(p_1, p_2)$ we have to check whether the edges to be deleted, i.e. $(\pi_{p_1}, \pi_{p_1+1})$ and $(\pi_{p_2}, \pi_{p_2+1})$, are contained in the tabu list. However, for such multi-attribute moves there are different ways to define the tabu criterion: a move may be classified tabu when at least one of the attributes

**Algorithm 1** Generic tabu search heuristic.

TabuSearch($s$, TabuMemory):

initialize TabuMemory
**while** ( stopping criterion not fulfilled )
  $s'$ = BestAdmissibleNeighbour( $s$, Neighbourhood, TabuMemory )
  TabuMemory.add( move$(s, s')$ )
  $s = s'$
  TabuMemory.add( $s$ )
  **if** ( escape triggered by TabuMemory )
    perform a diversifying move

of this move is contained in the tabu list, or when both are in the list. For our purpose both criteria are investigated.

### 3.2.2 Strict tabu search

The basic idea of strict tabu search is to provide necessity and sufficiency with respect to the idea of not revisiting any previously visited solution. So we have to prohibit a move to a neighbour if and only if this neighbour has already been visited during the previous part of the search. This may be done by alternative mechanisms: by exploiting logical interdependencies between the sequence of moves performed throughout the search process, as realized by the reverse elimination method, or by storing information about all solutions visited so far. This may be done either exactly or approximately by, e.g., the use of a hash code (cf. [18]). As the hash code of two different solutions may be the same whenever a so-called collision occurs, moves might be unnecessarily set tabu in some cases.

Here we do not use the reverse elimination method, but store the full permutation $\Pi = (\pi_1, \ldots, \pi_m)$ or a hash code $\sum_{i=1}^{m} r_i \pi_i$ (for a vector $(r_1, \ldots, r_m)$ of pseudo-random integers) of each solution visited. Each entry in this trajectory based memory is attributed by the iteration when the corresponding solution was visited the last time and by the frequency indicating how often this solution has been visited.

### 3.2.3 Reactive tabu search

Reactive tabu search is a modification of the static tabu criterion as the tabu duration is appropriately adapted throughout the search (cf. [19]). This is done by using a trajectory based memory as described in the previous subsection. Here we start with a tabu duration $d$ of 1 and increase it every time a solution has been repeated. The

increase of $d$ is defined by $d := \min\{\max\{d+1, d \times 1.1\}, u\}$, given an upper bound $u$ (i.e., the size of the neighbourhood minus 1 to guarantee at least one admissible move). If there has been no repetition for some iterations we decrease it appropriately to $\max\{\min\{d-1, d \times 0.9\}, 1\}$. We use a static tabu list as described in Section 3.2.1 with a trajectory memory by hash codes as described in Section 3.2.2.

# 4 Computational results

## 4.1 Problem data

We applied the heuristics described above for the larger instances of the problem sets described and used in [10]. The data represent heuristically determined solutions (patterns) of randomly generated cutting stock problems. The parameter $v$ represents the relative size of the largest length of an order compared to the standard length (cf. [10]). So a small $v$ leads to a higher density of the resulting matrices. There are four problem sets for $m = 50$ ($v = 0.25$, $v = 0.5$, $v = 0.75$, $v = 1$) with 100 problem instances each, and four problem sets for $m = 60$ with 100 instances for $v = 0.25$ and $v = 0.5$, 98 instances for $v = 0.75$ and 99 instances for $v = 1$ (summing up to 797 problem instances). For all problem instances $m$ is close to $n$. The frequencies $\alpha_i$ are all set to one.

Optimal solutions for the test sets are not available. Defining $\tilde{c}_{ij} = 1$ if $c_{ij} > 0$ (else 0), simple lower bounds for PSP-AOS may be computed as

$$AOS_{\mathrm{LB}} = \frac{1}{n} \sum_{j=1}^{n} \left( \sum_{i=1}^{m} \tilde{c}_{ij} \alpha_i - 1 \right),$$

and for PSP-SOS as

$$SOS_{\mathrm{LB}} = \max \left\{ \sum_{i=1}^{m} \tilde{c}_{ij} \mid j = 1, \ldots, n \right\}.$$

Unfortunately, for the problem sets considered here the ratio of the objective function value of the best solutions found (see below) to the computed simple lower bounds is about two to four, so we do not report these lower bounds here.

## 4.2 Implementation

The implementation was made in C++ using a framework with generic components for heuristic optimization. The different types of problems and different methods are incorporated in a unifying way without any fine tuning. This provides a way for a fair comparison of different heuristics by controlled and unbiased experiments.

The computations were performed on a PentiumII/266. To allow a comparison with the computing times of [15, 10], the speed ratio between a PentiumII/266 and a i486DX2/66 (as used in [10]) may be estimated for our purpose by 10, as our own tests have shown.

## 4.3 Results

In the following we summarize some results of the application of different heuristics on the eight problem sets described above. All results given in the tables below are average values over 100 (or 98 or 99) problem instances. $Z$ denotes the average objective function value, $\delta\%$ denotes the distance percent to the best average objective function value reached, $T$ denotes the average CPU time in seconds. With $s$ we denote the applied starting solution: identity permutation (s=id) in accordance with the numbering of the patterns or the solution determined by the cheapest insertion (s=ci) or the cheapest insertion of worst pattern (s=ciw) heuristic.

Table 1 shows the results of the application of both construction heuristics and steepest descent (2-opt). The cheapest insertion of worst pattern heuristic achieves significantly better results than the cheapest insertion heuristic, this supports the reasoning given in Section 2.1. Though, the solution quality of these construction heuristics is far from those achieved by the improvement methods. The moderate computing times of the construction heuristics and steepest descent support the efficiency of the algorithms given in Section 2. As tests have shown, methods with randomized algorithms like simulated annealing and tabu search with random escape moves do not necessarily require good starting solutions, whereas the solution quality of deterministic methods may increase significantly when using relatively good starting solutions, as is exemplified in Table 1 for steepest descent.

| | | **Chins** | | | **Chins worst** | | | **2-opt** | | | **2-opt** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | (ci) | | | (ciw) | | | s=ci | | | s=ciw | | |
| $m$ | $v$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ |
| 50 | 0.25 | 19.69 | 40.2 | *0.2* | 18.06 | 28.6 | *0.2* | 14.93 | 6.3 | *1.0* | 14.84 | 5.7 | *0.8* |
| 50 | 0.50 | 17.65 | 42.0 | *0.2* | 16.14 | 29.8 | *0.2* | 13.30 | 7.0 | *0.9* | 13.17 | 6.0 | *0.7* |
| 50 | 0.75 | 8.54 | 75.4 | *0.2* | 7.08 | 45.4 | *0.2* | 5.30 | 8.8 | *0.6* | 5.24 | 7.6 | *0.5* |
| 50 | 1.00 | 2.53 | 83.3 | *0.2* | 1.93 | 39.9 | *0.2* | 1.47 | 6.5 | *0.3* | 1.46 | 5.8 | *0.2* |
| 60 | 0.25 | 23.44 | 42.8 | *0.4* | 21.75 | 32.5 | *0.4* | 17.64 | 7.5 | *2.3* | 17.50 | 6.6 | *2.9* |
| 60 | 0.50 | 21.03 | 43.9 | *0.4* | 19.06 | 30.5 | *0.4* | 15.61 | 6.8 | *1.9* | 15.53 | 6.3 | *1.6* |
| 60 | 0.75 | 9.45 | 77.3 | *0.4* | 7.97 | 49.5 | *0.4* | 5.78 | 8.4 | *1.1* | 5.86 | 9.9 | *0.9* |
| 60 | 1.00 | 2.90 | 100.0 | *0.4* | 2.09 | 44.1 | *0.4* | 1.53 | 5.5 | *0.5* | 1.56 | 7.6 | *0.4* |

Table 1: Results for PSP-AOS.

To facilitate a comparison with the computational results of [10] we included their results (3-opt and simulated annealing) in the first columns of Table 2 with computation times scaled down by a factor of ten. Due to the high computation times of

| | | **3-opt [10]** | | | **SA [10]** | | | **SA** | | | **TSS** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | s=id | | | s=ciw | | |
| $m$ | $v$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ |
| 50 | 0.25 | 14.66 | 4.4 | *166.2* | 14.73 | 4.9 | *6.1* | 14.29 | 1.8 | *17.1* | 14.61 | 4.1 | *34.2* |
| 50 | 0.50 | 13.04 | 4.9 | *158.5* | 13.12 | 5.6 | *5.8* | 12.68 | 2.0 | *16.6* | 12.94 | 4.1 | *32.6* |
| 50 | 0.75 | 5.21 | 7.0 | *177.0* | 5.39 | 10.7 | *5.4* | 5.01 | 2.9 | *10.8* | 5.14 | 5.5 | *32.0* |
| 50 | 1.00 | 1.47 | 6.5 | *63.0* | 1.57 | 13.8 | *4.1* | 1.42 | 2.9 | *7.3* | 1.42 | 2.9 | *26.8* |
| 60 | 0.25 | – | – | – | 17.27 | 5.2 | *13.2* | 16.85 | 2.7 | *21.6* | 16.98 | 3.5 | *55.3* |
| 60 | 0.50 | – | – | – | 15.42 | 5.5 | *5.5* | 14.96 | 2.4 | *21.7* | 15.20 | 4.0 | *50.5* |
| 60 | 0.75 | – | – | – | 5.86 | 9.9 | *10.8* | 5.56 | 4.3 | *13.3* | 5.65 | 6.0 | *46.7* |
| 60 | 1.00 | – | – | – | 1.68 | 15.9 | *8.8* | 1.49 | 2.8 | *8.7* | 1.53 | 5.5 | *39.6* |

Table 2: Results for PSP-AOS.

| | | **TSTA** | | | **TSTAE** | | | **TSRE** | | | **TSRE5000** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | s=ciw | | | s=ciw | | | s=ciw | | | s=ciw | | |
| $m$ | $v$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ |
| 50 | 0.25 | 14.70 | 4.70 | *15.94* | 14.25 | 1.5 | *15.2* | 14.11 | 0.5 | *17.3* | 14.04 | 0.0 | *142.6* |
| 50 | 0.50 | 13.07 | 5.15 | *14.96* | 12.58 | 1.2 | *14.3* | 12.49 | 0.5 | *16.1* | 12.43 | 0.0 | *79.5* |
| 50 | 0.75 | 5.16 | 5.95 | *11.65* | 4.96 | 1.8 | *11.1* | 4.90 | 0.6 | *11.9* | 4.87 | 0.0 | *58.6* |
| 50 | 1.00 | 1.45 | 5.07 | *7.59* | 1.40 | 1.4 | *7.3* | 1.39 | 0.7 | *7.2* | 1.38 | 0.0 | *35.1* |
| 60 | 0.25 | 17.46 | 6.40 | *29.25* | 16.70 | 1.8 | *28.3* | 16.53 | 0.7 | *32.2* | 16.41 | 0.0 | *159.5* |
| 60 | 0.50 | 15.47 | 5.89 | *26.17* | 14.87 | 1.8 | *25.3* | 14.71 | 0.7 | *28.9* | 14.61 | 0.0 | *142.6* |
| 60 | 0.75 | 5.79 | 8.63 | *19.10* | 5.43 | 1.9 | *18.5* | 5.38 | 0.9 | *20.1* | 5.33 | 0.0 | *98.2* |
| 60 | 1.00 | 1.56 | 7.59 | *12.48* | 1.47 | 1.4 | *12.3* | 1.45 | 0.0 | *12.2* | 1.45 | 0.0 | *58.9* |

Table 3: Results for PSP-AOS.

3-opt, Foerster and Wäscher [10] do not provide the respective results for the problem instances with $m = 60$. Simulated annealing provides a significant improvement concerning solution quality compared to steepest descent. Our simulated annealing implementation was applied for 1,000,000 iterations, which resulted in significantly better results than those of [10]. This might be based on the more efficient neighbourhood evaluation used here which facilitated the execution of more iterations.

For static tabu search (TSS, last column of Table 2) we used a tabu duration of 50 for 1,000 iterations; moves are classified tabu when both attributes of this move are contained in the tabu list (cf. Section 3.2.1). Table 3 shows the results of the application of advanced tabu search methods: strict tabu search by approximate trajectory (TSTA) for 1,000 iterations, strict tabu search by approximate trajectory including five random escape moves (TSTAE) after every 100 iterations for 1,000 iterations, reactive tabu search including five random escape moves (TSRE) after every 100 iterations for 1,000 and 5,000 iterations. The solution quality of the simple tabu search approaches is not satisfactory. The primary reason for this might be the inability to diversify the search into unexplored regions of the search space as the neighbourhood generally includes $O(m^2)$ solutions with minor changes of the objective function value.

On the other hand, the results show the superiority of tabu search methods with the incorporation of simple diversification steps by random escape moves. Especially the reactive tabu search with escape moves generated very good solutions, while taking moderate running times.

The Tables 4, 5 and 6 show the respective results for PSP-SOS. We used for all computations the modified evaluation function as described in Section 2.4, as the simple evaluation, by the change of the objective function value, resulted in clearly worse results. The simulated annealing implementation was applied for 1,000,000 iterations, the tabu search algorithms for 1,000 iterations (with the exception of the last column of Table 6, that shows the reference results of 5,000 reactive tabu search iterations with escape moves). Concerning the comparison of the different methods, these results primarily support the findings for PSP-AOS. Due to the additional case c to be considered (cf. Section 2.4) the running times for PSP-AOS are generally higher than those of PSP-SOS.

To summarize the computational experiments, reactive tabu search with the inclusion of escape moves leads to overall best results. Simple tabu search approaches by itself are not competitive, even the application of simulated annealing leads to better results at an average. So we may conclude that diversification has been the crucial component of the heuristic strategies considered here.

| | | **Chins** | | | **Chins worst** | | | **2-opt** | | | **2-opt** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | (ci) | | | (ciw) | | | s=ci | | | s=ciw | | |
| $m$ | $v$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ |
| 50 | 0.25 | 27.32 | 34.85 | *0.5* | 25.95 | 28.08 | *0.8* | 21.30 | 5.13 | *4.8* | 21.27 | 4.99 | *4.1* |
| 50 | 0.50 | 24.44 | 32.11 | *0.5* | 24.15 | 30.54 | *0.8* | 19.60 | 5.95 | *4.0* | 19.68 | 6.38 | *3.8* |
| 50 | 0.75 | 14.13 | 62.04 | *0.5* | 12.14 | 39.22 | *0.8* | 9.31 | 6.77 | *2.7* | 9.48 | 8.72 | *2.4* |
| 50 | 1.00 | 7.55 | 91.62 | *0.5* | 4.89 | 24.11 | *0.8* | 4.16 | 5.58 | *1.4* | 4.10 | 4.06 | *1.1* |
| 60 | 0.25 | 31.73 | 37.36 | *1.0* | 29.94 | 29.61 | *1.7* | 24.45 | 5.84 | *12.3* | 24.59 | 6.45 | *10.8* |
| 60 | 0.50 | 29.55 | 36.24 | *1.0* | 28.60 | 31.86 | *1.7* | 23.12 | 6.59 | *9.7* | 23.19 | 6.92 | *8.6* |
| 60 | 0.75 | 15.69 | 63.78 | *1.0* | 13.83 | 44.36 | *1.7* | 10.38 | 8.35 | *5.9* | 10.39 | 8.46 | *5.3* |
| 60 | 1.00 | 8.48 | 104.34 | *1.0* | 5.20 | 25.30 | *1.7* | 4.34 | 4.58 | *3.1* | 4.26 | 2.65 | *2.2* |

Table 4: Results for PSP-SOS.

# 5 Conclusions

We presented the application of various heuristics for different types of pattern sequencing problems. Although the neighbourhood evaluation is rather expensive, the considered heuristics may be implemented with satisfying efficiency. The computational results achieved (regarding solution quality and running times) are superior to

| | | **SA** | | | **TSS** | | | **TSTA** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | s=id | | | s=ciw | | | s=ciw | | |
| $m$ | $v$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ |
| 50 | 0.25 | 20.75 | 2.42 | *47.1* | 21.32 | 5.23 | *98.3* | 21.19 | 4.59 | *67.3* |
| 50 | 0.50 | 19.00 | 2.70 | *47.7* | 19.79 | 6.97 | *95.6* | 19.49 | 5.35 | *65.1* |
| 50 | 0.75 | 9.10 | 4.36 | *51.7* | 9.37 | 7.45 | *98.8* | 9.41 | 7.91 | *65.5* |
| 50 | 1.00 | 4.00 | 1.52 | *72.0* | 4.03 | 2.28 | *56.2* | 4.09 | 3.81 | *37.8* |
| 60 | 0.25 | 23.89 | 3.42 | *60.6* | 24.76 | 7.19 | *164.0* | 24.58 | 6.41 | *127.6* |
| 60 | 0.50 | 22.42 | 3.37 | *61.9* | 23.23 | 7.10 | *151.2* | 22.98 | 5.95 | *115.8* |
| 60 | 0.75 | 10.29 | 7.41 | *63.7* | 10.39 | 8.46 | *152.4* | 10.26 | 7.10 | *95.7* |
| 60 | 1.00 | 4.31 | 3.86 | *87.4* | 4.24 | 2.17 | *91.7* | 4.26 | 2.65 | *63.8* |

Table 5: Results for PSP-SOS.

| | | **TSTAE** | | | **TSRE** | | | **TSRE5000** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | s=ciw | | | s=ciw | | | s=ciw | | |
| $m$ | $v$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ | $Z$ | $\delta\%$ | $T$ |
| 50 | 0.25 | 20.45 | 0.94 | *62.2* | 20.36 | 0.49 | *57.5* | 20.26 | 0.00 | *288.3* |
| 50 | 0.50 | 18.69 | 1.03 | *60.0* | 18.64 | 0.76 | *56.1* | 18.50 | 0.00 | *282.2* |
| 50 | 0.75 | 8.80 | 0.92 | *58.5* | 8.81 | 1.03 | *53.8* | 8.72 | 0.00 | *300.1* |
| 50 | 1.00 | 3.94 | 0.00 | *32.7* | 3.95 | 0.25 | *29.5* | 3.94 | 0.00 | *147.2* |
| 60 | 0.25 | 23.34 | 1.04 | *115.3* | 23.30 | 0.87 | *99.3* | 23.10 | 0.00 | *531.4* |
| 60 | 0.50 | 21.99 | 1.38 | *106.7* | 21.93 | 1.11 | *97.7* | 21.69 | 0.00 | *521.6* |
| 60 | 0.75 | 9.68 | 1.04 | *99.7* | 9.67 | 0.94 | *91.4* | 9.58 | 0.00 | *446.0* |
| 60 | 1.00 | 4.15 | 0.00 | *56.5* | 4.18 | 0.72 | *51.4* | 4.15 | 0.00 | *250.3* |

Table 6: Results for PSP-SOS.

those presented in the literature. This may rule out the exclusive application of simple priority rules as examined in [13, 11, 15].

Further work should include theoretical research, e.g., concerning the conjectures given in [1] about the relationships of PSP-SOS with related problems.

In addition, our findings may be incorporated and tested within general frameworks for heuristic methods where efficient ideas for software reuse are the main focus. Respective research may underline the generality of our research as various problem types may be subsumed under the heading of pattern sequencing.

# References


[1] H.H. Yanasse. On a pattern sequencing problem to minimize the maximum number of open stacks. *European Journal of Operational Research* **100**, 454–463 (1997).

[2] O.B.G. Madsen. An application of travelling-salesman routines to solve pattern-allocation problems in the glass industry. *Journal of the Operational Research*

*Society* **39**, 249–256 (1988).

[3] T.C.E. Cheng, J. Diamond, and B.M.T. Lin. Optimal scheduling in film production to minimize talent hold cost. *Journal of Optimization Theory and Applications* **79**, 197–206 (1993).

[4] A.-L. Nordström and S. Tufekci. A genetic algorithm for the talent scheduling problem. *Computers & Operations Research* **21**, 927–940 (1994).

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

[6] S.B. Deutsch and J.J. Martin. An ordering algorithm for analysis of data items. *Operations Research* **19**, 1350–1362 (1971).

[7] W.T. McCormick Jr., P.J. Schweitzer, and T.W. White. Problem decomposition and data reorganization by a clustering technique. *Operations Research* **20**, 993–1009 (1972).

[8] P.M. Morse. Optimal linear ordering of information items. *Operations Research* **20**, 741–751 (1972).

[9] J.A. Hoffer and D.G. Severance. The use of cluster analysis in physical database design. In *Proceedings of the First International Conference on Very Large Data Bases (VLDB)*, IEEE Computer Society, 69–86 (1975).

[10] H. Foerster and G. Wäscher. Simulated annealing for the order spread minimization problem in sequencing cutting patterns. *European Journal of Operational Research*, to appear (1998).

[11] B.J. Yuen. Improved heuristics for sequencing cutting patterns. *European Journal of Operational Research* **87**, 57–64 (1995).

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, New York (1979).

[13] B.J. Yuen. Heuristics for sequencing cutting patterns. *European Journal of Operational Research* **55**, 183–190 (1991).

[14] B.J. Yuen and K.V. Richardson. Establishing the optimality of sequencing heuristics for cutting stock problems. *European Journal of Operational Research* **84**, 590–598 (1995).

[15] H. Foerster. *Fixkosten- und Reihenfolgeprobleme in der Zuschnittplanung.* PhD thesis, Technische Universität Braunschweig (1998).

[16] K.A. Dowsland. Simulated Annealing. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, 20–69, Blackwell Scientific Publications, Oxford (1993).

[17] F. Glover and M. Laguna. *Tabu Search.* Kluwer, Boston et al. (1997).

[18] D.L. Woodruff and E. Zemel. Hashing vectors for tabu search. *Annals of Operations Research* **41**, 123–138 (1993).

[19] R. Battiti. Reactive search: Toward self-tuning heuristics. In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, *Modern Heuristic Search Methods*, 61–83, Wiley, Chichester (1996).

# Appendix

**Algorithm 2** Evaluation of inserting pattern $i$ between positions $p-1$ and $p$ for PSP-AOS.

$\Delta = 0$
**for** ( $j$=1; $j \leq n$; ++$j$ )
  **if** ( ( first[$j$] == 0 ) **and** ( $c_{ij} > 0$ ) )
    $\Delta = \Delta + \alpha_i - 1$
  **else if** ( ( $p \leq$ first[$j$] ) **and** ( $c_{ij} > 0$ ) )
    $\Delta = \Delta + \alpha_i$
    **for** ( $l$=$p$; $l <$ first[$j$]; ++$l$ )
      $\Delta = \Delta + \alpha_{\pi_l}$
  **else if** ( ( $p >$ first[$j$] ) **and** ( $p \leq$ last[$j$] ) )
    $\Delta = \Delta + \alpha_i$
  **else if** ( ( $p >$ last[$j$] ) **and** ( $c_{ij} > 0$ ) )
    $\Delta = \Delta + \alpha_i$
    **for** ( $l$=p-1; $l >$ last[$j$]; --$l$ )
      $\Delta = \Delta + \alpha_{\pi_l}$

**Algorithm 3** Evaluation of inserting pattern $i$ between positions $p-1$ and $p$ for PSP-SOS.

```
adaptedOpen = open
adaptedOpen[0] = 0
for ( j=1; j ≤ n; ++j )
    if ( ( first[j] == 0 ) and ( c_ij > 0 ) )
        ++adaptedOpen[0]
    else if ( ( p ≤ first[j] ) and ( c_ij > 0 ) )
        ++adaptedOpen[0]
        for ( l = p; l < first[j]; ++l )
            ++adaptedOpen[l]
    else if ( ( p > first[j] ) and ( p ≤ last[j] ) )
        ++adaptedOpen[0]
    else if ( ( p > last[j] ) and ( c_ij > 0 ) )
        ++adaptedOpen[0]
        for ( l=p-1; l > last[j]; --l )
            ++adaptedOpen[l]
Δ = max{adaptedOpen[0,...]} − max{open[1,...]}
```

**Algorithm 4** Evaluation of a 2-exchange neighbour for PSP-AOS.

```
Δ = 0
for ( j=1; j ≤ n; ++j )
    if ( ( p_1 < first[j] ) and ( p_2 ≥ first[j] ) and ( p_2 < last[j] ) )
        // case b
        for ( i=p_1+1; i < first[j]; ++i )
            Δ = Δ − α_{π_i}
        for ( i=p_2; not c_{π_i,j}; --i )
            Δ = Δ + α_{π_i}
    else if ( ( p_1 ≥ first[j] ) and ( p_1 < last[j] ) and ( p_2 ≥ last[j] ) )
        // case e
        for ( i=p_2; i > last[j]; --i )
            Δ = Δ − α_{π_i}
        for ( i=p_1+1; not c_{π_i,j}; ++i )
            Δ = Δ + α_{π_i}
Δ = Δ / n
```

**Algorithm 5** Basic evaluation of a 2-exchange neighbour for PSP-SOS.

adaptedOpen = open
**for** ( $j$=1; $j \leq n$; ++$j$ )
  **if** ( ( $p_1 <$ first[$j$] ) **and** ( $p_2 \geq$ first[$j$] ) **and** ( $p_2 <$ last[$j$] ) )
    // case b
    $i = p_2$
    **while** ( **not** $c_{\pi_i,j} > 0$ )
      --$i$
    **if** ( ($p_1$+1)+($p_2 - i$) < first[$j$] )
      **for** ( k=($p_1$+1)+($p_2 - i$); $k <$ first[$j$]; ++$k$ )
        ++adaptedOpen[$k$]
    **else for** ( $k$=first[$j$]; $k < (p_1 + 1) + (p_2 - i)$; ++$k$ )
        --adaptedOpen[$k$]
  **else if** ( ( $p_1 <$ first[$j$] ) **and** ( $p_2 \geq$ last[$j$] ) )
    // case c
    **if** ( (first[$j$] − ($p_1$+1)) > ( $p_2$ − last[$j$] ) )  // first and last would go down
      **for** ( $k$ = ($p_1$+1)+($p_2$ − last[$j$]); $k <$ first[$j$]; ++k )
        ++adaptedOpen[$k$]
      **for** ( $k$ = last[$j$]; $k > p_2$ − (first[$j$] − ($p_1$+1)); --k )
        --adaptedOpen[$k$]
    **else**  // first and last would go up
      **for** ( $k$ = first[$j$]; $k <$ ($p_1$+1)+($p_2$ − last[$j$]); ++$k$ )
        --adaptedOpen[$k$]
      **for** ( $k = p_2$ − (first[$j$] − ($p_1$+1)); $k >$ last[$j$]; --$k$ )
        ++adaptedOpen[$k$]
  **else if** ( ( $p_1 \geq$ first[$j$] ) **and** ( $p_1 <$ last[$j$] ) **and** ( $p_2 \geq$ last[$j$] ) )
    // case e
    $i = p_1 + 1$
    **while** ( **not** $c_{\pi_i,j} > 0$ )
      ++$i$
    **if** ( $p_2 - (i - p_1 - 1) >$ last[$j$] )
      **for** ( $k = p_2 - (i - p_1 - 1)$; $k >$ last[$j$]; --$k$ )
        ++adaptedOpen[$k$]
    **else for** ( $k$ = last[$j$]; $k > p_2 - (i - p_1 - 1)$; --$k$ )
        --adaptedOpen[$k$]
$\Delta = \max\{\text{adaptedOpen}[1], \ldots, \text{adaptedOpen}[m]\} - \max\{\text{open}[1], \ldots, \text{open}[m]\}$