ELSEVIER International Journal of Approximate Reasoning 19 (1998) 73–90

INTERNATIONAL JOURNAL OF APPROXIMATE REASONING

Arranging program statements for locality on the basis of neighbourhood preferences

Claudia Leopold [1]

Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, 07740 Jena, Germany

Received 1 March 1997; accepted 1 November 1997

Abstract

The gradual property of computer programs, that their successive operations preferably access data from the same memory block, is called locality. The paper deals with locality optimization, more specifically with the sequencing aspect that N operations are to be brought into sequence such that locality is maximized. We assume to be given a matrix $D = [D_{ij}]$ of neighbourhood preferences, where entry D_{ij} is the smaller the higher the expected gain in locality when arranging operations o_i and o_j closely. The gain is supposed to have been estimated from so far accumulated but still incomplete knowledge of an overall locality optimization process. Our task consists in finding a sequencing function $T: \{o_1 \dots o_N\} \to [1 \dots N] \subseteq \mathbb{R}$ that assigns to each operation a real time at which it will be approximately carried out. The motivation for T mapping into reals instead of integers is to transfer more knowledge on the certainty of operation ordering decisions into the next step of the overall locality optimization process. The goal for T consists in minimizing an objective function that was empirically designed to approximately quantify the intuitive meaning of the degree of locality. In addition, T has to spread the values $T(o_i)$ quite evenly over the interval $[1 \dots N]$. We suggest a heuristic algorithm that approximately solves the problem, and report on experiments with the algorithm and several variants of it. Briefly, the algorithm starts with a random sequencing that is iteratively improved, by alternatingly moving each $T(o_i)$ in the direction of the value that minimizes the objective function for fixed $T(o_j)(j \neq i)$, and spreading the $T(o_i)$ over $[1 \dots N]$. Experimental results indicate that our algorithm is efficient and reasonably accurate.

[1] E-mail: claudia@minet.uni-jena.de.

0888-613X/98/$19.00
PII: S0888-613X(98)00015-2

Keywords: Sequencing; Scheduling under uncertainty; Data locality; Iterative improvement; Descent algorithms; Objective function methods

1. Introduction

Computers typically have a memory hierarchy with limited amount of fast memory. Programs run faster if operations that access the same datum or the same memory block (e.g. cache line) are arranged closely. The reason is that only sufficiently close operations can reuse data brought into fast memory, otherwise the data are overwritten in the meantime. The gradual property of computer programs, that successive operations preferably access data from the same memory block, is called *locality*. Programs tend to run the faster the higher their degree of locality is.

In [1], an iterative method to automatic data locality optimization was suggested that will be summarized in the following. The method combines two complementary [2] sources for locality optimization:

- data assignment, i.e., assigning the data to memory blocks, and
- sequencing, i.e., ordering the operations of the program.

The input program is given as a *set* of statement *instances* and a *set* of data *elements*. Note that loops are unrolled into statement instances. In the following, we will use the notions statement instances, statements and operations interchangeably. The statements are characterized by their access behaviour, i.e., we know which statement will access which data, and, for conditional statements, also with which likelihood. The aim is to find a data assignment and sequencing to maximize the locality of the program.

The method tackles this complicated optimization problem iteratively, by alternatingly refining the data assignment and the sequencing, starting with a data assignment step. Only the sequencing is dealt with in the present paper. Taking a closer look at the iterative process, the first step has to find a data assignment *without any knowledge on the operation sequencing*. Hence, it assigns to a common memory block those data that are accessed together in some statement. It cannot, however, take into account that data accessed in successive statements should also be assigned to a common memory block, though this is of equal importance, i.e., the data assignment step must take a decision based on incomplete knowledge. Analogously, the sequencing step that follows must decide for a sequencing on the basis of an only preliminary data assignment, and so on. Thus, each decision is of necessity arbitrary to a certain degree. If arbitrariness has led to a wrong decision in one step, it will often imply a wrong decision in the next step, too. Hence, there is a high danger of ending in a local optimum only.

To weaken this danger, the iterative method permits fuzziness in the decisions taken in intermediate steps. The fuzzy decisions convey more information into

the next step than crisp decisions and can be more easily corrected in successive steps. Hence, they promise a decreasing danger of ending in a local optimum only. Throughout the iterative process, the degree of fuzziness is decreased, since at the very end we need a crisp data assignment and statement ordering.

For the data assignment, fuzziness has the form of fuzzy membership values of data in blocks. They are conceptually fuzzy, since, for different data, the importance for being stored in a particular set is different: Those data that are frequently used with other data from the set have a high membership value, those that are rarely used with other data from the set have a low membership value. On the other hand, ultimately we need a crisp data assignment that respects the requirement of a fixed block size but comes close to the fuzzy membership values. Hence, the membership values can also be understood as preliminary approximations of the probability that the datum will finally be assigned to the corresponding memory block.

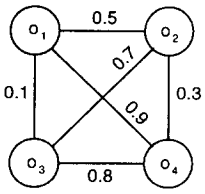
Prior to the sequencing phase, the membership values are transformed into a neighbourhood preference matrix D that reflects the degree of preference of statement pairs for being placed close together. D is obtained via $D = SD \circ (DB \circ' DB^{-1}) \circ SD^{-1}$, where SD and DB are matrices with $SD[s,d]$ being an estimation of the likelihood that statement s accesses datum d, and $DB[d,b]$ being the membership value of datum d in block b. The composition operators $\circ$ and $\circ'$ are probabilistic, justified by the probabilistic view of the membership values and the fact that the entries of SD are probabilities. $\circ'$ is chosen such that $(DB \circ' DB^{-1})[d,d']$ is an approximation of the probability that d and d' will be finally assigned to the same memory block (taking the membership values as probabilities). $\circ$ is matrix multiplication with subsequent scaling into $[0,1]$ such that $D[o_i,o_j]$ reflects the expected number of reuses between statements o_i and o_j.

In the output of the sequencing, fuzziness is expressed via the concept of a real-valued sequencing function introduced in the following: Usually, in sequencing, the task is to bring N operations $o_1 \ldots o_N$ into sequence such that the result is a one-to-one function $S: \{o_1 \ldots o_N\} \rightarrow \{1 \ldots N\}$ with $\{1 \ldots N\} \subseteq \mathbb{Z}$. $S(o_i) = t$ denotes that operation o_i will be executed at time t. Note that we assume all operations to have equal lengths. The concept of a *real-valued sequencing function*, in contrast, consists in that the result is a function $T: \{o_1 \ldots o_N\} \rightarrow [1 \ldots N] \subseteq \mathbb{R}$. The meaning of T is twofold: First, T represents a sequencing, namely the same sequencing as the integer-valued function S that is obtained from T by ordering T's values (i.e., $S(o_i) = k$ iff $T(o_i)$ is the kth largest value in the range of T). S will also be referred to as the sequencing *corresponding* to T. Second, T indicates how certain the decision for the relative ordering of operations in the sequencing is. If it is a clear advantage to place some operation o_i before some other operation o_j, then the distance between o_i and o_j is large, on the real scale. If it is only a minor advantage, then the distance is smaller.

Consider as an example the matrix

$$D = \begin{pmatrix} 0 & 0.5 & 0.1 & 0.9 \\ 0.5 & 0 & 0.7 & 0.3 \\ 0.1 & 0.7 & 0 & 0.8 \\ 0.9 & 0.3 & 0.8 & 0 \end{pmatrix},$$

which represents the following graph:



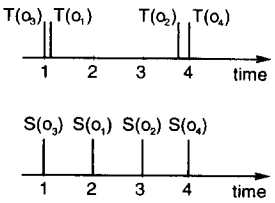
A good sequencing function, i.e., a function that places operations with high neighbourhood preference closely and spreads its values over the whole interval $[1 \ldots N]$ would e.g.be

$$T(o_1) = 1.04, \quad T(o_2) = 3.85, \quad T(o_3) = 1.0, \quad T(o_4) = 4.0,$$

The corresponding integer-valued sequencing is

$$S(o_1) = 2, \quad S(o_2) = 3, \quad S(o_3) = 1, \quad S(o_4) = 4$$

or, represented graphically,



It is to be seen that T does not only represent the order of operations, but it also reflects that the order of e.g. operations o_1 and o_2 is much more certain than that of operations o_1 and o_3. Hence, minor modifications of D may change the optimal order of operations o_1 and o_3, but they will not change the optimal order of operations o_1 and o_2.

The so far loose requirement of placing operations with high neighbourhood preference "closely" is quantified through the following objective function

$$f(N,T,D) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\left(\ln(\mid T(o_j) - T(o_i) \mid +1)\right)^2}{D_{ij}}.$$

The function is to be minimized under the constraint that T must spread its values quite evenly over $[1 \ldots N]$ (see Section 2).

This objective function approximately reflects the requirements of a good schedule: A large distance between operations o_i and o_j is penalized the more the higher the neighbourhood preference of o_i and o_j is. This is reflected by having $\mid T(o_j) - T(o_i) \mid$ in the numerator and D_{ij} in the denominator. The logarithmic scale causes decreases in distance to pay out more for moderately far away operations than for operations that are in any case too far away to permit reuse. The exponent 2 improves analytic accessibility. The choice of function f is still somewhat arbitrary in that other scalings could produce the same qualitative effect like the logarithmic scaling, and in that additional parameters like the base of the logarithm could adapt f to concrete values of the memory size. From design, our algorithm should be quite robust towards modifications of the objective function. This expectation was also confirmed by initial experimental results. Hence, the objective function f considered in this paper should be sufficiently representative.

The meaning of closeness is specified fuzzily, since the question if two statements are placed closely enough to permit reuse can, at compile time, often not be answered affirmatively. Reasons are cache effects (direct mapped or set-associative caches), conditional statements with different numbers of data accessed in the branches, and may be machine dependence.

The sequencing problem considered in the present paper is still simplified in that it does not include hard precedence constraints between operations that need to be respected when there are data dependencies between the statements of the input program.

In the formulation of the sequencing problem, we have, for simplicity, assumed the operations to be of equal length. It should not be a problem to generalize our algorithm to operations of different lengths. A straightforward solution could be to represent longer operations by several shorter operations with a very high degree of preference for being executed close to each other.

The paper is organized as follows. Section 2 gives a concise statement of the problem considered. In Section 3, we describe our basic algorithm and motivate its design. Briefly, the algorithm starts with a random sequencing that is iteratively improved, by alternatingly moving each $T(o_i)$ in the direction of the value that minimizes f for fixed $T(o_j)$ $(j \neq i)$, and spreading the $T(o_i)$ over $[1 \ldots N]$. Implementation details such as the choice of parameters are discussed in Section 4, where also several variants of the basic algorithm are introduced. Particularly interesting is a variant where not only steps that decrease but with some probability also steps that increase the objective function value are ac-

cepted (similar to simulated annealing). In Section 4, we furthermore explain two reference algorithms we have used in our experiments: an enumerative algorithm and a local search algorithm based on pairwise exchange of operations. Section 5 lists and discusses our experimental results. Section 6 overviews related work and Section 7 finishes with conclusions.

2. Formal statement of the problem

A set of N operations $o_1 \ldots o_N$ is to be brought into sequence. Input is an $N \times N$ matrix $D = [D_{ij}]$ with

$$0 \leqslant D_{ij} \leqslant 1 \quad (1 \leqslant i, j \leqslant N),$$

$$D_{ij} = D_{ji} \quad (1 \leqslant i, j \leqslant N),$$

$$D_{ij} = 0 \leftrightarrow i = j \quad (1 \leqslant i, j \leqslant N).$$

D is a dissimilarity matrix, i.e., entry D_{ij} is the smaller the higher the preference of operations o_i and o_j for being arranged closely. Further requirements on D are not imposed, in particular D need neither fulfill the triangle inequality nor be Euclidean (embeddable into Euclidean space with distances D_{ij} between points i and j).

The task is to find a sequencing function $T: \{o_1 \ldots o_N\} \to [1 \ldots N] \subseteq \mathbb{R}$ that minimizes, at least approximately, the objective function

$$f(N, T, D) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\left(\ln(|\, T(o_j) - T(o_i) \,| + 1)\right)^2}{D_{ij}}.$$

Additionally, T must spread its values quite evenly over $[1 \ldots N]$. We have experimented with several quantifications of this constraint, all requiring $|T(o_i) - S(o_i)|$ to be less than some bound which is e.g. a constant. Here, S denotes the sequencing corresponding to T, see Section 1. In addition, there must be operations o_i and o_j with $T(o_i) = 1$ and $T(o_j) = N$.

Sometimes, we will also consider an integer-valued version of the optimization problem. Here, the task is to directly find a function $S: \{o_1 \ldots o_N\} \to \{1 \ldots N\} \subseteq \mathbb{Z}$ that minimizes the objective function

$$f(N, S, D) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\left(\ln(|\, S(o_j) - S(o_i) \,| + 1)\right)^2}{D_{ij}}$$

(without constraints).

3. Our algorithm

The algorithm is heuristic and in essence an iterative descent method. It starts with a random sequencing function $T^{(0)}$ that, with uniform distribution over $[1 \ldots N]$, independently assigns to each operation o_i a random initial time $T^{(0)}(o_i)$. In each step k, the algorithm cycles through the values $T^{(k-1)}(o_i)$, replacing each $T^{(k-1)}(o_i)$ by an intermediate value $T^{(k)}_{\text{help}}(o_i)$ such that $T^{(k)}_{\text{help}}(o_i)$ comes closer to the value that minimizes the objective function for fixed $T^{(k-1)}(o_j)$ $(j \neq i)$. After each cycle, the $T^{(k)}_{\text{help}}(o_i)$ are spread over $[1 \ldots N]$ to avoid that the values of the sequencing function move closer and closer together. The spread values are the new $T^{(k)}(o_i)$.

A necessary condition for the objective function to take a minimum at some $T^{(k)}_{\text{help}}(o_i)$ (for fixed $T^{(k-1)}(o_j)$ $(j \neq i)$) is that its derivative equals zero for this $T^{(k)}_{\text{help}}(o_i)$. Unfortunately, equating the derivative to zero leads, for our objective function, to an equation that is difficult to solve. Hence, we will use an approximation.

Let us first consider a simpler objective function

$$g(N,T,D) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{(T(o_j) - T(o_i))^2}{D_{ij}}.$$

Then,

$$\frac{\mathrm{dg}}{\mathrm{d}T(o_i)} = \frac{\mathrm{d}}{\mathrm{d}T(o_i)} \left(\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{(T(o_j) - T(o_i))^2}{D_{ij}} \right) = 2 \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{T(o_i)}{D_{ij}} - 2 \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{T(o_j)}{D_{ij}}.$$

Hence, g has a local extremum at

$$T(o_i) = \frac{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{T(o_j)}{D_{ij}}}{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{1}{D_{ij}}}.$$

It must be the global minimum, since $g(.)$ obviously takes smaller values for arguments from inside the interval

$$\left[\min_{1 \leq j \leq N} \{T(o_j)\} \ldots \max_{1 \leq j \leq N} \{T(o_j)\} \right]$$

that for other arguments $T(o_i)$. Thus, g must possess a global minimum within this interval. The above given local extremum is the only candidate.

Coming back to the minimization of f, the above calculation is helpful if we consider the following scaling of the time scale (scaled values are marked by overlining):

$$\bar{t} = \ln(|\, t - T^{(k-1)}(o_i) \,| + 1) \operatorname{sign}(t - T^{(k-1)}(o_i))$$

for the currently to be updated $T^{(k-1)}(o_i)$ and any time $t \in \mathbb{R}^+$. ($\mathrm{sign}(x) = 1$ if $x > 0$ and $\mathrm{sign}(x) = -1$ otherwise). Hence $\overline{T^{(k-1)}(o_i)} = 0$. With the new scaling, f applied to the values of step $k-1$ appears as

$$f(N, T^{(k-1)}, D) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{\left(\overline{T^{(k-1)}(o_j)} - \overline{T^{(k-1)}(o_i)}\right)^2}{D_{ij}},$$

which has the same structure as g and hence, under the condition of fixed $\overline{T^{(k-1)}(o_i)}$, takes a global minimum if $\overline{T^{(k-1)}(o_i)}$ is replaced by

$$\overline{T^{(k)}_{\mathrm{help}}(o_i)} = \frac{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{\overline{T^{(k-1)}(o_j)}}{D_{ij}}}{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{1}{D_{ij}}}. \tag{1}$$

Though $T^{(k)}_{\mathrm{help}}(o_i)$, the value corresponding to $\overline{T^{(k)}_{\mathrm{help}}(o_i)}$ on the original scale, is in general not the global minimum of f, it approximates the minimum at least in so far as that it indicates in which direction from $T^{(k-1)}(o_i)$ the new value $T^{(k)}_{\mathrm{help}}(o_i)$ should be located and gives a rough estimation on how far away it should be located. The approximation seems to be sufficient for the purpose of using it in our iterative approximation algorithm, as finally indicated by the experimental results. A more exact approximation at this point could be a source for further improvement of the algorithm.

We still have to invert the scaling, to locate $T^{(k)}_{\mathrm{help}}(o_i)$ on the original scale:

$$\overline{T^{(k)}_{\mathrm{help}}(o_i)} = \ln\left(\left|T^{(k)}_{\mathrm{help}}(o_i) - T^{(k-1)}(o_i)\right| + 1\right) \mathrm{sign}\left(T^{(k)}_{\mathrm{help}}(o_i) - T^{(k-1)}(o_i)\right)$$

implies

$$\left|\overline{T^{(k)}_{\mathrm{help}}(o_i)}\right| = \ln\left(\left|T^{(k)}_{\mathrm{help}}(o_i) - T^{(k-1)}(o_i)\right| + 1\right)$$

and

$$\mathrm{sign}\left(\overline{T^{(k)}_{\mathrm{help}}(o_i)}\right) = \mathrm{sign}\left(T^{(k)}_{\mathrm{help}}(o_i) - T^{(k-1)}(o_i)\right).$$

Hence,

$$\left|T^{(k)}_{\mathrm{help}}(o_i) - T^{(k-1)}(o_i)\right| = \mathrm{e}^{\left|\overline{T^{(k)}_{\mathrm{help}}(o_i)}\right|} - 1$$

and consequently

$$T^{(k)}_{\mathrm{help}}(o_i) = \left(\mathrm{e}^{\left|\overline{T^{(k)}_{\mathrm{help}}(o_i)}\right|} - 1\right) \mathrm{sign}\left(\overline{T^{(k)}_{\mathrm{help}}(o_i)}\right) + T^{(k-1)}(o_i). \tag{2}$$

After each cycle of replacing the $T^{(k-1)}(o_i)$ by $T^{(k)}_{\mathrm{help}}(o_i)$ according to Eq. (2), the values are spread over $[1 \ldots N]$, using a simple proportional spreading scheme.

In the simplest case, we fix the two outermost values at 1 and N, respectively, and proportionally adapt the remaining values. Hence,

$$T^{(k)}(o_i) = \frac{(N-1)\left(T^{(k)}_{\text{help}}(o_i) - \min_j\{T^{(k)}_{\text{help}}(o_j)\}\right)}{\max_j\{T^{(k)}_{\text{help}}(o_j)\} - \min_j\{T^{(k)}_{\text{help}}(o_j)\}} + 1. \tag{3}$$

Despite the spreading, it may still happen that the values tend to cluster in some few points. In this case, we do not only fix the outermost values but also some symmetrically chosen values between. Their number will be denoted by N_{fix}. We have experimented with several variants of choosing N_{fix}, see Section 4. If e.g. $N_{\text{fix}} = 4$, then we would fix the smallest, the $\lfloor N/3 \rfloor$-largest, the $\lfloor 2N/3 \rfloor$-largest and the largest $T^{(k)}_{\text{help}}(o_i)$ values at $T^{(k)}(o_i) = 1$, $\lfloor N/3 \rfloor$, $\lfloor 2N/3 \rfloor$ and N, respectively, and proportionally adapt the remaining values.

Simply applying Eqs. (2) and (3) alternatingly leads to a process that typically does not converge, but shows an oscillating behaviour similar to that observed for gradient descent methods when a too large step size has been chosen. For this reason, we introduce an additional parameter $B \in [0 \ldots 1]$, the binding strength of the old values, and replace Eq. (2) by

$$T^{(k)}_{\text{help}}(o_i) = (1-B)\, T^{(k)}_{\text{help}(2)}(o_i) + B\, T^{(k-1)}(o_i), \tag{4}$$

where $T^{(k)}_{\text{help}(2)}(o_i)$ is the value calculated according to Eq. (2). B is initialized with a small N-dependent value, and slowly increased in the course of the iterative process. Details on the increasing scheme will be given in Section 4. The effect of B is that at the beginning of the iterative process, the values of the sequencing function may vary heavily between iterations. Hence, in a sense, the solution space is scanned coarsely and the objective function typically quickly approaches a value close to the optimum. This value is later fine-tuned when a large B permits only minor adjustments of the sequencing function.

In summary, our algorithm starts with a random function $T^{(0)}$ that is iteratively improved. Each iteration step carries out one cycle through the values $T^{(k-1)}(o_i)$, replacing them by $T^{(k)}_{\text{help}}(o_i)$ according to Eq. (4). Then the values are spread according to Eq. (3) into $T^{(k)}(o_i)$. Convergence of the algorithm is enforced via the parameter B. The algorithm stops when B has reached some threshold. There are several variants of our algorithm that will be detailed in the next section.

4. Implementation and reference algorithms

4.1. Implementation details

Random number generator: To initialitize the function $T^{(0)}$ as well as for random decisions in the increasing scheme for B (see below), and also to generate

test graphs for our experiments, we need a random number generator with uniform distribution. We used the generator RAN2 from [3] for all purposes, since it avoids sequential correlations and is quite fast.

Sequential vs. parallel update of $T(o_i)$: In the basic variant described in Section 3, the updating step Eq. (4) (see Eq. (1)) refers to the old values $T^{(k-1)}(o_j)$. This has the advantage that potentially all $T^{(k)}_{\text{help}}(o_i)$ can be evaluated in parallel, hence we will refer to this version as parallel update. Alternatively, we could in Eq. (1) replace $T^{(k-1)}(o_j)$ by $T^{(k)}_{\text{help}}(o_j)$ for those $T^{(k)}_{\text{help}}(o_j)$ that have already been evaluated before. This will be referred to as sequential update.

Determining N_{fix}: Except of some few test runs where we have omitted the spreading entirely until some bound was violated, we have always started with $N_{\text{fix}} = 2$, and incremented N_{fix} by one if the $T^{(k)}_{\text{help}}(o_i)$ values clustered too heavily. Referring to the integer-valued function $S^{(k)}_{\text{help}}$ corresponding to $T^{(k)}_{\text{help}}$, we considered three definitions of what it means to cluster too heavily:

$$|T^{(k)}_{\text{help}}(o_i) - S^{(k)}_{\text{help}}(o_i)| > c_1\,(N/N_{\text{fix}}) \quad \text{for some } i.,$$

$$|T^{(k)}_{\text{help}}(o_i) - S^{(k)}_{\text{help}}(o_i)| > c_2 N \quad \text{for some } i,$$

$$|T^{(k)}_{\text{help}}(o_i) - S^{(k)}_{\text{help}}(o_i)| > c_3 \quad \text{for some } i,$$

Basic increasing scheme for B: Here, B is initialized to $B^{(0)} = 1/N$. Let $B^{(k)}$ denote the value of B used for step k of our algorithm. Then we have either $B^{(k)} = B^{(k-1)}$ or $B^{(k)} = (B^{(k-1)} + 1)/2$. We have $B^{(k)} = (B^{(k-1)} + 1)/2$ iff $f(N, T^{(k-1)}, D) > f(N, T^{(k-2)}, D)$ except if N_{fix} was incremented in step $k-1$ and not in step $k-2$. Otherwise $B^{(k)} = B^{(k-1)}$. The scheme has been derived from manual experience with the algorithm.

Randomized increasing scheme for B: This scheme was inspired by simulated annealing (see e.g. [4]). To make the algorithm more flexible, we accept not only steps that decrease but with some probability also steps that increase the value of the objective function. B is again initialized to $B^{(0)} = 1/N$. If

$$f(N, T^{(k-1)}, D) \leqslant f(N, T^{(k-2)}, D)$$

then we always keep $B^{(k)} = B^{(k-1)}$ (independent on N_{fix}). If

$$f(N, T^{(k-1)}, D) > f(N, T^{(k-2)}, D)$$

then we keep with some probability $B^{(k)} = B^{(k-1)}$, and otherwise set $B^{(k)} = (B^{(k-1)} + 1)/2$. In accordance with simulated annealing, the probability is determined as

$$\exp\left\{ -\left(\frac{f(N, T^{(k-1)}, D)}{\min\limits_{i=1\ldots k-1} \{f(N, T^{(i)}, D)\}} - 1 \right) \left(\frac{1}{F^{(k-1)}} \right) \right\},$$

where $F^{(k-1)}$ is a parameter called temperature. We are using the ratio instead of the difference between $f(N, T^{(k-1)}, D)$ and $\min\limits_{i=1\ldots k-1} \{f(N, T^{(i)}, D)\}$ since objec-

tive function values may vary heavily between different input graphs; the ratio gives independence from absolute values without requiring to adapt the temperature parameter. $F^{(k)}$ has been determined from manual experience with the algorithm as $F^{(k)} = 1.5(1 - B^{(k)})^2$, where the squaring has the effect that the duration of the steps does not grow too much for large k. We restrict the maximum number of iterations carried out with the same $B^{(k)}$ to 100, to guarantee termination also in case of convergence, and to guarantee bounds on the running time of the algorithm.

Last value vs. best value: The objective function value of the final sequencing may be worse than that of some intermediate sequencing. This is e.g. due to the fact that B is updated on the basis of T, not of S, in the integer-valued version of the optimization problem. We let the algorithm either return the objective function value of the final S or the best objective function value encountered so far. The former is referred to as last value, the latter as best value. Clearly, the best value can never be worse than the last value. Returning the last value saves the time to evaluate intermediate $S^{(k)}$ and their objective function values $f(N, S^{(k)}, D)$.

Last-intermediate vs. best-intermediate: While with the random increasing scheme, after increasing B, we always reset $T^{(k-1)}$ to the best so far sequencing function, for the basic increasing scheme we have tried both keeping $T^{(k-1)}$ at the function determined in step $k-1$, and resetting $T^{(k-1)}$ to the best so far function. We refer to the former case as last-intermediate and to the latter as best-intermediate variant.

4.2. Reference algorithms

As it was only for the integer-valued version of the optimization problem that we found different but well-established heuristics for similar optimization problems in the literature, we took this version as a basis for evaluation. Our algorithm is applied to the integer-valued version by first running the algorithm as usual to find a function T, and then returning the function S that corresponds to T. The quality of S is an indicator of the quality of T, because T is an intermediate result. In the following, two reference algorithms are described.

Enumerative algorithm: A trivial algorithm for the integer-valued version of our problem is to systematically generate all the $N!$ candidates for S, determine $f(N, S, D)$ and return the minimum. The algorithm is guaranteed to yield the exact optimum, but takes exponential time and is prohibitively slow already for about $N > 12$.

Pairwise exchange heuristic: This is a simple local search heuristic, similar to the 2-opt edge exchange heuristic for the travelling salesman problem [5]. The algorithm starts with $S: \{o_1 \ldots o_N\} \to \{1 \ldots N\}$ being a random one-to-one function. Then, it repeatedly picks two operations o_i and o_j and exchanges

$S(o_i)$ and $S(o_j)$ if the exchange decreases the objective function value by some minimum amount. The algorithm stops if there is no exchange possible anymore. The algorithm has freedom in the choice of the $S(o_i) - S(o_j)$ pair to be exchanged next. Taking solution quality and computation time into consideration, after some experimentation, we decided to consider the $S(o_i)$ in cyclic order, choose the best $S(o_j)$ for the current $S(o_i)$ and exchange $S(o_i)$ and $S(o_j)$ if there is any gain in the objective function value. This variant is used as the reference algorithm.

5. Experimental results

We have run several variants of our algorithm and the reference algorithms on typically 500 randomly generated matrices D. Tables 1 and 2 list some characteristic results. Where nothing else is indicated, our algorithm was run with the following parameters: parallel update of $T(o_i)$, basic increasing scheme for B, $c_1 = 0.9$ in the criterion for determining N_{fix}, best value is returned, last intermediate value is used, algorithm stops when $B > 0.95$.

Table 1 compares our algorithm with the pairwise exchange heuristic. The given values are ratios between the objective function values achieved with our algorithm and with the pairwise exchange heuristic. More specifically, Table 1 lists the average, maximum and minimum ratios obtained over 500 runs. The upper part of the table refers to the above given standard variant of our algorithm, for the lower part, the randomized increasing scheme was used (with remaining parameters as in the standard variant). It is to be seen that the objective function values of our algorithm are off those of the pairwise exchange heuristic by about 10%. The randomized scheme achieves somewhat better results than the basic scheme.

Table 1
Ratios between objective function values

N	5	7	8	10	20	50	100
Our algorithm (standard) / pairwise exchange							
Avg.ratio	1.09	1.09	1.08	1.09	1.09	1.10	1.12
Max.ratio	3.7	2.5	2.0	1.4	1.6	1.4	1.4
Min.ratio	0.88	0.88	0.83	0.89	0.90	0.95	1.0
Our algorithm (rand.increasing scheme) / pairwise exchange							
Avg.ratio	1.06	1.06	1.06	1.07	1.06	1.08	1.10
Max.ratio	2.1	2.6	1.9	3.3	2.6	3.1	1.9
Min.ratio	0.84	0.77	0.82	0.85	0.86	0.89	0.98

Table 2
Ratios between objective function values

N	5	7	8	10	20	50	100
Our algorithm (standard) / our algorithm (stop for $B > 0.99999$)							
Avg.ratio	1.05	1.02	1.01	1.0	1.0	1.0	1.0
Our algorithm (last value returned) / our algorithm (standard)							
Avg.ratio	1.02	1.02	1.02	1.02	1.01	1.00	1.00
Our algorithm (sequential update) / our algorithm (standard)							
Avg.ratio	0.99	1.03	1.03	1.03	1.02	1.02	1.01
Our algorithm (standard) / our algorithm (best intermediate)							
Avg.ratio	1.01	0.99	0.99	0.98	0.97	0.97	0.98
Pairwise exchange / enumerative							
N	5	7	8	9	10	11	12
Avg.ratio	1.01	1.02	1.03	1.04	1.04	1.09	1.05
Max.ratio	1.3	1.2	1.2	1.3	1.2	1.2	1.07
# Test runs	500	500	500	500	150	3	3

Table 2 shows that another stopping criterion cannot improve the solution quality significantly, and also that there is no significant gain in using the best value. Furthermore, Table 2 shows that with sequential update or best intermediate, results were slightly worse than with the standard variant. This may be due to the fact that the remaining parameters were tuned for the standard variant. Nevertheless, the experiments seem to indicate that not much gain can be expected from these variations. Table 2 also compares the pairwise exchange heuristic with the enumerative algorithm. Though this was practicable only for small values of N, the results suggest that the pairwise exchange heuristic is less than 10% off the optimum, and that the deviation is rather independent on N.

We have also run the algorithm on some application-specific graphs, generated from a program for multiplying 2×2 matrices ($N = 8$) with several degrees of fuzziness in the data assignment, and observed similar results as with the random graphs. Objective function values have been 0–8% off the optimum if the degree of fuzziness was high in D, they have been up to 14% off the optimum if D was close to a 0-1-matrix.

We tried initializing N_{fix} with 1, but, though it improved the solution quality for about $N = 3 \ldots 5$, it was disadvantageous for larger N. We experimented with $c_2 = 0.2$ and $c_3 = 2$, and observed about the same results as with $c_1 = 0.9$, at least for $N \geqslant 10$. The algorithm was quite robust towards the choice of the parameter c_1: varying c_1 between 0.6 and 1.0 changed the objec-

Table 3
Influence of multiple initializations

k	1	2	3	4	5
Our alg. (standard with k initializations) / pairwise exch., N = 50					
Avg.ratio	1.10	1.08	1.06	1.06	1.05
Our alg. (rand. scheme with k initializations) / pairw. exch., N = 50					
Avg.ratio	1.08	1.05	1.045	1.040	1.037

tive function value by less than 1%, setting $c_1 = 1.3$ decreased the objective function value by 2%, for the standard variant.

It pays off to run our algorithm with multiple initializations as shown in Table 3. This result was to be expected, since the algorithm typically converges to a local but not necessarily to a global optimum. Multiple initializations cover different local optima.

Fig. 1 gives time measurements referring to straightforward codings of the algorithms in Modula-2 on a DECstation 5000. The time measurements have been averaged over 500 test runs, except for the slow programs where it has been less test runs.

So far, the pairwise exchange heuristic had been coded with solution quality as the primary aim. We also tried running the pairwise exchange heuristic for the same amount of time as our algorithm (just stopping the program when time was over), and observed the pairwise exchange heuristic to be still slightly superior to our algorithm. Table 4 gives some results.

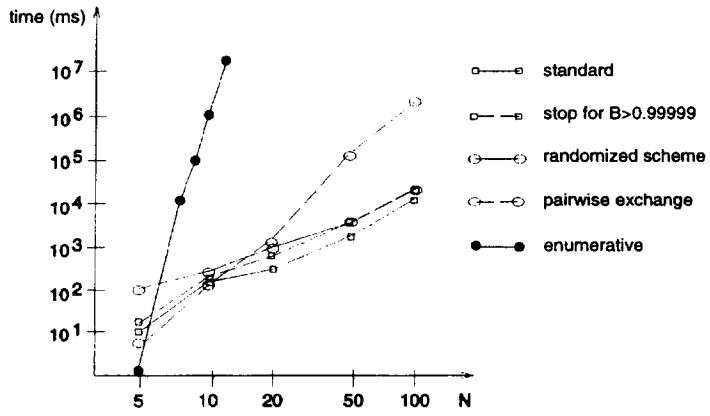


Fig. 1. CPU time for several algorithms.

Table 4
Ratios of objective function values for equal running times

N	5	7	8	10	20	50	100
Our algorithm(standard) / pairwise exch.(same time)							
Avg.ratio	1.07	1.08	1.06	1.07	1.04	1.05	1.07

Hence, for the integer-valued version of our problem, the pairwise exchange heuristic leads to slightly better sequencings than our algorithm, but the difference is rather small and perhaps even due to current parameter settings.

The situation is different for the real-valued version of our problem, however, that cannot even be handled with the pairwise exchange heuristic. To nevertheless evaluate the pairwise exchange heuristic in this situation, we have run a combination between the pairwise exchange heuristic and our algorithm (standard variant) against our algorithm alone. In the combination, our algorithm was initialized with the result of the pairwise exchange heuristic, and it started with a larger $B^{(0)}$ than usual. Our algorithm was not only much faster than the combination (the combination has about the same running time as the pairwise exchange heuristic), but it also achieved slightly better objective function values. They are given in Table 5.

While for integer results, the recorded ratios have been relatively constant over different test runs (rarely over 20% deviation for $N > 10$), the ratios fluctuated more heavily for real results. For our parameter settings, objective function values obtained for the real-valued version have been about 60% of those obtained for the integer-valued version. For the real-valued version, objective function values obtained with the randomized scheme were about 10% worse than those obtained with the standard variant. This may be due to current parameter settings.

We also did some initial experiments with other objective functions. Some results are listed in Table 6. We expect that the results can be improved by adapting the parameters to the respective objective function, but we have not yet tried this in the experiments.

Table 5
Comparison of the combination against our standard algorithm

N	5	7	8	10	20	50	100
Our algorithm (standard) / combination, $B^{(0)} = 0.7$							
Avg.ratio	1.2	1.07	1.07	1.07	0.92	0.98	0.95
Our algorithm (standard) / combination, $B^{(0)} = 0.5$							
Avg.ratio	1.2	1.08	1.09	1.14	0.96	0.92	0.90

Table 6
Results with other objective functions

N	10	20	50	100
Our algorithm (standard)/pairwise exch (same time)				
$f(N,T,D)=\sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{\lvert T(o_j)-T(o_i)\rvert^2}{D_{ij}}$				
Avg.ratio	1.16	1.13	1.14	1.18
Time in ms	2×10^1	1×10^2	4×10^2	1×10^3
$f(N,T,D)=\sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{\sqrt{\lvert T(o_j)-T(o_i)\rvert}}{D_{ij}}$				
Avg.ratio	1.06	1.08	1.10	1.11
time in ms	2×10^2	6×10^2	2×10^3	1×10^4

To summarize, the results indicate that our algorithm typically achieves objective function values within 20% of the optimum. Though we have done some experimentation, we expect that the quality of the algorithm can be further improved by fine-tuning the parameters. The algorithm exhibits a trade-off between solution quality and computation time that can be shifted with other settings. The experiments indicate that our algorithm is sufficiently efficient for the applications we have in mind and reasonably accurate. Initial measurements indicate that it will have similar properties for other objective functions.

6. Related work

The topic of this paper is different from other data locality optimization research which typically aims at regular loop structures and considers a restricted set of candidate transformations only (see e.g. [6], or [1] for a discussion). There is related work from other areas, though.

The sequencing problem considered in this paper can be understood as a scheduling problem [7] with a single processor and a special type of constraints. The constraints are the (typically conflicting) neighbourhood preferences. The aim of the sequencing problem corresponds to finding compromises between these constraints. In this sense, the constraints are comparable to the fuzzy constraints of [8].

Other related work is multidimensional scaling [9,10] that aims at embedding a dissimilarity matrix into Euclidean space such that the distances between data points approximate the dissimilarities. For a one-dimensional space, it is similar to our problem, except that the objective function favours both related data points to be placed close to each other and unrelated data points to be placed far away. Our problem is restricted to the former.

Our algorithm closely resembles the centroid and annealing heuristics developed in the context of visualizing semantic nets [11] in three-dimensional space.

Differences result mainly from that we require the algorithm to spread the T values evenly, whereas [11] requires them to be discriminable.

Our algorithm was inspired by Fuzzy Clustering [12]. The problem of fuzzy clustering resembles our problem in that a (dis)similarity matrix is given, and objects are to be arranged with the aim of realizing neighbourhood preferences. In fuzzy clustering, the objects are to be arranged into groups, whereas in our problem they are to be arranged on a one-dimensional scale. Both cases use an objective function that is minimized in an iterative process, where, assuming some parameters of the arrangement to be fixed, other parameters are set to optimum (or close to optimum) values, determined with the help of a derivative.

The algorithm also shares similarity with the Elastic Net approach to the travelling salesman problem in that real values are moving under the influence of several forces (in our case, attractive forces of the preferred neighbours and the binding force of the old value), and the relative strengths of the forces change in the course of the optimization process.

In general, our algorithm resembles the well-known gradient descent optimization methods in that it is iterative and determines the next iterate with the help of the partial derivatives. Our optimization problem could alternatively be tackled with gradient descent methods. We preferred the present approach because of its comparatively low computational expense for determining the next iterate, and because of the special situation arising from the constraints.

The idea of relaxing the requirement of integer-valued schedules, i.e., considering T instead of S, was also used in e.g. [13]. There it was used for another purpose, though (being able to apply techniques from nonsmooth optimization to the schedule construction, not to get a real-valued result).

Increasing B in the iterative process to first coarsely and later finely search for an optimum is similar in spirit to simulated annealing [14] or deterministic annealing [4].

Finally, our problem is related to the Minimum Linear Arrangement and Minimum Bandwidth problems [15], except that we are considering weighted graphs and have a different objective function.

7. Conclusions

In this paper, we have defined a sequencing problem where statements should be arranged with the aim of fulfilling neighbourhood preferences. The result is a sequencing function that assigns to each operation a *real* time at which it will be approximately carried out. The problem is a problem of scheduling under uncertainty where fuzziness/imprecision pertains to:

- the matrix D of neighbourhood preferences,
- the representation of the result (T instead of S), and
- the meaning of closeness.

We have introduced and evaluated an algorithm to approximately solve the stated problem. Experimental results indicate that the algorithm is efficient and reasonably accurate. Though we have already experimented with several parameter settings, we expect that the quality of the algorithm can be further improved by fine-tuning the parameters. An important open problem is to include hard precedence constraints, that represent data dependencies, into the algorithm.

References

[1] C. Leopold, A fuzzy approach to automatic data locality optimization, Proceedings of ACM Symposium on Applied Computing, 1996, 515–518.
[2] M. Cierniak, W. Li, Unifying data and control transformations for distributed shared memory machines, Proceedings of ACM Symposium on Programming Language Design and Implementation, 1995, pp. 205–217.
[3] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes, Cambridge University Press, London, 1986.
[4] S.C. Shapiro, Encyclopedia of Artificial Intelligence, Wiley, New York, 1992.
[5] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds.), The Traveling Salesman Problem A Guided Tour of Combinatorial Optimization, Wiley, New York, 1995.
[6] D.F. Bacon, S.L. Graham, O.J. Sharp, Compiler transformations for high-performance computing, ACM Computing Surveys 26 (4) (1994) 345–420.
[7] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, J. Węglarz, Scheduling Computer and Manufacturing Processes, Springer, Berlin, 1996.
[8] W. Slany, Scheduling as a fuzzy multiple criteria optimization problem, Fuzzy Sets and Systems 78 (1996) 197 222.
[9] T.F. Cox, M.A.A. Cox, Multidimensional Scaling, Chapman & Hall, London, 1994.
[10] T. Hofmann, J. Buhmann, Multidimensional scaling and data clustering, in: Advances in Neural Information Processing Systems 7, Morgan Kaufmann, Los Altos, CA, 1995, pp. 104–111.
[11] K.M. Fairchild, S.E. Poltrock, G.W. Furnas, SemNet: Three-Dimensional Graphic Representations of Large Knowledge Bases, Lawrence Erlbaum, London, 1988.
[12] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, New York, 1981.
[13] W. Achtziger, K.-H. Zimmermann, Optimal polynomial schedules: An approach via nonsmooth optimization, Proceedings of the Workshop on Approximate Reasoning in Scheduling, ICSC Press, 4–10, 1997.
[14] E. Gurewitz, K. Rose, G.C. Fox, Constrained clustering as an optimization method, IEEE Trans. on Pattern Analysis and Machine Intelligence 15 (8) (1993) 785–794.
[15] P. Crescenzi, V. Kann, A compendium of NP optimization problems, Technical Report SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Università di Roma La Sapienza, 1995.