Sequencing with Neighbourhood Preferences

Claudia Leopold
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena
07740 Jena, Germany
claudia@minet.uni-jena.de

Keywords: sequencing, data locality, iterative improvement, objective function methods

ABSTRACT

We consider the problem of bringing N operations into sequence, with the aim of realizing neighbourhood preferences. The preferences are given by a dissimilarity matrix D, where entry D(i,j) is the smaller the higher the preference of operations i and j for being scheduled close to each other. The result is a sequencing function $T : \{1 \ldots N\} \rightarrow [1 \ldots N]$ that should minimze the objective function

$$f = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{(\ln(\mid T(j) - T(i) \mid +1))^2}{D(i,j)}$$

and also spread the values $T(i)$ quite evenly over the real interval $[1 \ldots N]$. 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 $T(i)$ in the direction of the value that minimizes f for fixed $T(j)(j \neq i)$, and spreading the $T(i)$ over $[1 \ldots N]$. An integer-valued version of our algorithm is evaluated against an enumerative algorithm and against a different heuristic algorithm. Experimental results indicate that our algorithm is efficient and reasonably accurate.

1 Introduction

1.1 Problem Statement

A set of N operations is to be brought into sequence, with the aim of realizing neighbourhood preferences given by an $N \times N$ matrix D with

- $0 \leq D(i,j) \leq 1 \qquad (1 \leq i,j \leq N)$
- $D(i,j) = D(j,i) \qquad (1 \leq i,j \leq N)$, and
- $D(i,j) = 0 \;\leftrightarrow\; i = j \qquad (1 \leq i,j \leq N)$

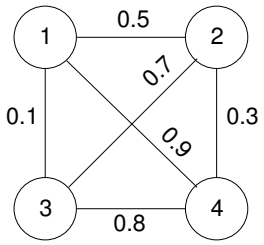
D is a dissimilarity matrix, i.e., entry $D(i,j)$ is the smaller the higher the preference of operations i and j to be arranged close to each other in the sequence. The meaning of 'close' will later be quantified through an objective function. Further requirements on D are not imposed, in particular D need neither fulfill the triangle inequality nor be euclidean (embeddable into euclidean space with distances $D(i,j)$ between points i and j).

The result is a sequencing function $T : \{1 \ldots N\} \rightarrow [1 \ldots N]$ that assigns to each operation a real time at which it will be approximately carried out. From T, a permutation $S : \{1 \ldots N\} \rightarrow \{1 \ldots N\}$ can be easily derived, by ordering the real numbers. The reason why we focus on T instead of S will be elaborated in section 1.2. Briefly, T conveys more information than S since it does not only represent the order of operations but it also reflects how certain this order is.

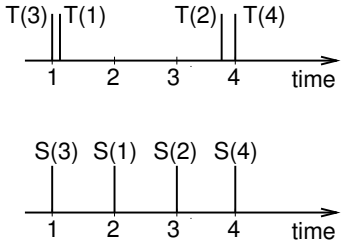
Consider as an example the matrix

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

or, represented as a graph,



A good sequencing function would be $T(1) = 1.04$, $T(2) = 3.85$, $T(3) = 1.0$, $T(4) = 4.0$, with derived sequencing $S(1) = 2$, $S(2) = 3$, $S(3) = 1$, $S(4) = 4$:



We see that T does not only represent the order of operations, but it also reflects that the order of e.g. operations 1 and 2 is much more certain than that of operations 1 and 3. Hence, minor modifications of D may change the optimal order of operations 1 and 3, but they will not change the optimal order of operations 1 and 2.

The quality of T is evaluated using the objective function

$$f = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{(\ln(\mid T(j) - T(i) \mid +1))^2}{D(i,j)}$$

that should be minimized. In addition, T is required to spread the values quite evenly over $[1 \ldots N]$. We have experimented with several quantifications of this requirement, all requiring $\mid T(i) - S(i) \mid$ to be less than some bound.

1.2 Motivation of the Problem

The background of the problem is in data locality optimization for computer programs. Since computers typically have a memory hierarchy with limited amount of fast memory, programs run the faster, the closer they arrange operations accessing the same datum or the same memory block (e.g. cache line). The reason is that only sufficiently close operations can reuse data once brought into fast memory, otherwise they will be overwritten in the meantime.

In [8], an iterative approach to automatic data locality optimization was suggested, where the assignment of data to memory blocks and the sequencing of operations are alternatingly refined. Current assignments are represented by fuzzy membership values, to convey more information into the next sequencing step than with crisp values, and to pronounce that the assignment is only preliminary and based on vague knowledge about the operation order. Analogously, using the real-valued sequencing function T, we can convey more information into the next data assignment step than by using the integer-valued sequencing S.

In the problem statement, both the neighbourhood preferences and the meaning of closeness have been specified fuzzily. For neighbourhood preferences, the reason is that the preferences are based on fuzzy membership values for the data assignment, they depend on how many data can be reused, and they depend on the execution frequencies of the statements. The meaning of closeness is fuzzy, since we are considering compile time where knowledge is limited because of cache effects, input dependent control structures, and maybe machine independence.

The function f approximately reflects the requirements to a good schedule: Large distances between operations i and j in the schedule increase the objective function value the more the closer i and j should be scheduled according to D (reflected by having $\mid T(j) - T(i) \mid$ in the numerator and $D(i,j)$ in the denominator). Because of the logarithmic scale, decreases in distance pay out more for reasonably 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.

1.3 Overview of the Paper

In section 2, 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 $T(i)$ in the direction of the value that minimizes f for fixed $T(j)$ $(j \neq i)$, and

spreading the $T(i)$ over $[1 \ldots N]$. Implementation details such as the choice of parameters are discussed in section 3, 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 accepted (similar to simulated annealing). In section 3, we furthermore explain two reference algorithms we have used in our experiments: an enumerative algorithm and a heuristic algorithm based on pairwise exchange of operations. Section 4 lists and discusses our experimental results. The paper finishes with some remarks about related work and conclusions in section 5.

2 Our Algorithm

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

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

Let us first consider a simpler objective function:

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

Then,

$$\begin{aligned} \frac{\mathrm{d}g}{\mathrm{d}T(i)} &= \frac{\mathrm{d}}{\mathrm{d}T(i)} \left(\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{(T(j) - T(i))^2}{D(i,j)} \right) \\ &= 2 \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{T(i)}{D(i,j)} - 2 \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{T(j)}{D(i,j)} \end{aligned}$$

Hence, g has a local extremum at

$$T(i) = \frac{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{T(j)}{D(i,j)}}{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{1}{D(i,j)}}$$

It must be the global minimum, since $g(.)$ obviously takes smaller values for arguments from inside the interval $[\min_{1 \leq j \leq N} \{T(j)\} \ldots \max_{1 \leq j \leq N} \{T(j)\}]$ than for other arguments, 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):

$$\overline{t} = \ln(\mid t - T^{(k-1)}(i) \mid +1) \cdot \mathrm{sign}(t - T^{(k-1)}(i))$$

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

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

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

$$\overline{T^{(k)}_{\text{help}}(i)} = \frac{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{\overline{T^{(k-1)}(j)}}{D(i,j)}}{\sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{1}{D(i,j)}}. \qquad (1)$$

Though $T^{(k)}_{\text{help}}(i)$, the value corresponding to $\overline{T^{(k)}_{\text{help}}(i)}$ 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 $T^{(k-1)}(i)$ the new value $T^{(k)}_{\text{help}}(i)$ 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.

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

$$\begin{aligned}\overline{T^{(k)}_{\text{help}}(i)} &= \ln\left(\left|T^{(k)}_{\text{help}}(i) - T^{(k-1)}(i)\right| + 1\right) \\ &\quad \cdot \text{sign}\left(T^{(k)}_{\text{help}}(i) - T^{(k-1)}(i)\right)\end{aligned}$$

implies

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

and

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

Hence,

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

and consequently

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

After each cycle of replacing the $T^{(k-1)}(i)$ by $T^{(k)}_{\text{help}}(i)$ according to (2), the values are spreaded 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,

$$T^{(k)}(i) = \frac{(N-1) \cdot \left(T^{(k)}_{\text{help}}(i) - \min_j\{T^{(k)}_{\text{help}}(j)\}\right)}{\max_j\{T^{(k)}_{\text{help}}(j)\} - \min_j\{T^{(k)}_{\text{help}}(j)\}} + 1 \quad (3)$$

according to the intercept theorem.

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 3). 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 $T^{(k)}_{\text{help}}(i)$ values at $T^{(k)}(i) = 1$, $\lfloor N/3 \rfloor$, $\lfloor 2N/3 \rfloor$ and N respectively.

Due to the effects of spreading and the approximation in the derivation of (2), simply applying (2) and (3) alternatingly leads to a process that often does not converge. For this reason, we introduce an additional parameter $B \in [0 \ldots 1]$, the binding strength of the old values, and replace (2) by

$$T^{(k)}_{\text{help}}(i) = (1-B) \cdot T^{(k)}_{\text{help}(2)}(i) + B \cdot T^{(k-1)}(i) \quad (4)$$

where $T^{(k)}_{\text{help}(2)}(i)$ is the value calculated according to (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 3. 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 close to optimum value. 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 variables $T^{(k-1)}(i)$, replacing them by $T^{(k)}_{\text{help}}(i)$ according to (4). Then the values are spreaded according to (3) into $T^{(k)}(i)$. Additional parameters, in particular B, let the process converge. The process stops when B has reached some threshold. There are several variants of our algorithm that will be detailed in the next section.

3 Implementation and Reference Algorithms

3.1 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 [9] for all purposes, since it avoids sequential correlations and is quite fast.

Sequential vs. Parallel Update of T(i): In the basic variant described in section 2, the updating step (4) (see (1)) refers to the old values $T^{(k-1)}(j)$. This has the advantage that potentially all $T^{(k)}_{\text{help}}(i)$ can be determined in parallel, hence we will refer to this version as parallel update. Alternatively, we could in (1) replace $T^{(k-1)}(j)$ by $T^{(k)}_{\text{help}}(j)$ for those $T^{(k)}_{\text{help}}(j)$ that have already been evaluated before. This will be referred to as sequential update.

Determining $\mathbf{N_{fix}}$: Except 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 $T^{(k)}_{\text{help}}(i)$ values clustered too heavily. Referring to the permutation $S^{(k)}_{\text{help}}$ derived from $T^{(k)}_{\text{help}}$, we considered three definitions of what it means to cluster too heavily:

- $|T^{(k)}_{\text{help}}(i) - S^{(k)}_{\text{help}}(i)| > f_1 \cdot (N/N_{\text{fix}})$, for some i,
- $|T^{(k)}_{\text{help}}(i) - S^{(k)}_{\text{help}}(i)| > f_2 N$, for some i, and
- $|T^{(k)}_{\text{help}}(i) - S^{(k)}_{\text{help}}(i)| > f_3$, for some i,

with parameters f_1, f_2 and f_3.

Basic Increasing Scheme for B: Here, B is initialized to $1/N$. It is increased in step k iff the objective function value increased from step $k-2$ to step $k-1$, except if N_{fix} was incremented in step $k-1$ and not in step $k-2$. Then, $B^{(k)}$ is set to $(B^{(k-1)} + 1)/2$, 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. [11]). 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 $1/N$. If the objective function value decreases from step $k-2$ to step $k-1$, we always keep $B^{(k)} = B^{(k-1)}$ (independent on N_{fix}). If the objective function value increases from step $k-2$ to step $k-1$, 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

$$e^{-(F^{(k-1)}/F^{\text{best}}-1)/T}$$

where $F^{(k-1)}$ and F^{best} denote the objective function value after step $k-1$ and the best so far objective function value respectively. T is a parameter called temperature. We are using the ratio instead of the difference between $F^{(k-1)}$ and F^{best} since objective function values may vary heavily between different input graphs; the ratio gives independence from absolute values without requiring to adapt T. The parameter T has been determined from manual experience with the algorithm as $T = 1.5 \cdot (1-B)^2$. We restrict the maximum number of iterations carried out with the same B to $M = 100$.

Real Result vs. Integer Result Though the algorithm has been designed to determine a sequencing function T, it can also be used to determine a permutation S by simply ordering T's values. The objective function value determined from T is referred to as real result, the objective function value determined from S is referred to as integer result (even though the objective function value itself will be real).

Last Value vs. Best Value In particular because B is updated on the basis of T even if we wish an integer result, the objective function value of the final S may be worse than that of some intermediate S. We let the algorithm either return the objective function value of the final S (referred to as last value) or the best objective function value encountered so far (referred to as best value). Even though the best value can never be worse than the last value, it may make sense to return the last value, since it saves the time to evaluate intermediate S and their corresponding objective function values.

While with the random increasing scheme, after increasing B, we always reset $T^{(k)}$ to the best so far sequencing function, for the basic increasing scheme we have tried both keeping $T^{(k)}$ at the function determined with the old B, and resetting $T^{(k)}$ to the best so far function. We refer to the former case as last-intermediate and to the later as best-intermediate variant. The former may have the advantage of being slightly faster.

3.2 Reference Algorithms

We evaluated our algorithm based on integer results. Since integer results are derived from real results, the quality of the integer results is an indicator of the quality of the real results. 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 their objective function values 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 heuristic algorithm, similar to the Kernighan-Lin heuristic for graph partitioning ([6]) or the 2-opt edge exchange heuristic for the travelling salesman problem ([7]). The algorithm starts with S being a random permutation of $\{1 \ldots N\}$. Then it repeatedly picks two indices i and j and exchanges $S(i)$ and $S(j)$ 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 $S(i)-S(j)$ pair to be exchanged next. Taking solution quality and computation time into consideration, after some experimentation, we decided for the variant of choosing the best pair for fixed $S(i)$ (considering the $S(i)$ in cyclic order), with exchange for any gain in objective function value, and used it in the reference algorithm.

4 Experimental results

We have run several variants of our algorithm and the reference algorithms on typically 500 randomly generated matrices D and recorded the ratios between the objective function values achieved with the algorithms. The following table lists some characteristic results, referring to statistics on the specified ratio.

variant$_1$/pairw.exch., with variant$_1$ = par.update, $f_1 = 0.9$, basic scheme, best value, $B <= 0.95$					
N	5	10	20	50	100
avg. ratio	1.14	1.09	1.09	1.10	1.12
max. ratio	3.7	1.4	1.6	1.4	1.4
min. ratio	0.88	0.89	0.90	0.95	1.0
variant$_2$/pairw.exch., with variant$_2$ = par.update, $f_1 = 0.9$, rand.scheme, best value, $B <= 0.95$					
N	5	10	20	50	100
avg. ratio	1.06	1.07	1.06	1.08	1.10
max. ratio	2.1	3.3	2.6	3.1	1.9
min. ratio	0.84	0.85	0.86	0.89	0.98
variant$_1$/variant$_3$, with variant$_3$ = par.update, $f_1 = 0.9$, basic scheme, best value, $B \leq 0.99999$					
N	5	10	20	50	100
avg. ratio	1.05	1.0	1.0	1.0	1.0
variant$_4$/variant$_1$, with variant$_4$ = par.update, $f_1 = 0.9$, basic scheme,last value, $B \leq 0.95$					
N	5	10	20	50	100
avg. ratio	1.00	1.02	1.01	1.00	1.00
pairw.exch./enumerative					
N	5	8	10	11	12
avg. ratio	1.01	1.03	1.04	1.09	1.05
max. ratio	1.3	1.2	1.2	1.2	1.07
# test runs	500	500	150	3	3

Many more results have been omitted for brevity. To summarize a few, we have run the algorithm on some application-specific graphs, generated from a program for multiplying 2×2 matrices with several degrees of fuzziness in the data assignment, and observed similar results as with the random graphs. Variants with serial update have been slightly worse (about 2 %) than variants with parallel update. Best-intermediate variants have not been better but even slightly worse (about 2%) than last-intermediate variants. Initializing N_{fix} with 1 improves the solution quality for about $N = 3 \ldots 5$, but is disadvantageous for other N. We experimented with $f_2 = 0.2$ and $f_3 = 2$, and observed about the same results as with $f_1 = 0.9$ (at least for $N \geq 10$).

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. In average, real results have been about 60% the amount of integer results. We compared real results of a combination where variant$_1$ was initialized with the result of the pairwise exchange algorithm to pure variant$_1$, and found the later to be superior by about 4%. Comparing the real result versions of variants 1 and 2, variant 1 was up to 10% better which may be due to current parameter settings.

The following time measurements refer to straightforward codings of the algorithms in Modula-2 on a DECstation 5000. The purpose of the time measurements was to compare the different algorithms, not to focus on absolute values. Time refers to CPU time in ms, averaged over 500 (or less, for the slow programs) test runs.

variant$_1$					
N	5	10	20	50	100
time	14	167	$5.8 \cdot 10^2$	$3.0 \cdot 10^3$	$1.1 \cdot 10^4$
variant$_2$					
N	5	10	20	50	100
time	130	368	$1.3 \cdot 10^3$	$5.6 \cdot 10^3$	$2.0 \cdot 10^4$
variant$_3$					
N	5	10	20	50	100
time	24	183	$7.1 \cdot 10^2$	$5.1 \cdot 10^3$	$2.7 \cdot 10^4$
pairw. exch.					
N	5	10	20	50	100
time	8	135	$2.7 \cdot 10^3$	$1.2 \cdot 10^5$	$2.2 \cdot 10^6$
enumerative					
N	5	8	9	10	11
time	0	9583	$1.1 \cdot 10^5$	$1.4 \cdot 10^6$	$1.8 \cdot 10^7$

It pays off to run our algorithm with multiple initializations as shown in the following table.

variant$_1$(k initializations) / pairw.exch., $N = 50$					
k	1	2	3	4	5
avg.ratio	1.10	1.08	1.06	1.06	1.05
variant$_2$(k initializations) / pairw.exch., $N = 50$					
k	1	2	3	4	5
avg.ratio	1.08	1.05	1.045	1.040	1.037

To summarize, the results indicate that the algorithm typically achieves objective function values within $10-20\%$ 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 tradeoff between solution quality and computation time that can be shifted with other settings. The time measurements clearly indicate that our algorithm is faster than the pairwise exchange heuristic, and is sufficiently efficient for the application we have in mind.

5 Related Work and Conclusions

In this paper, we have defined a sequencing problem, where operations should be arranged with the aim of fulfilling neighbourhood preferences. The result is a sequencing function that assigns to each operation a *real* time when it will be approximately carried out. We have introduced and evaluated an algorithm to approximately solve the problem, and experimented with several parameter settings. Future work will address the issue of including hard constraints on the operation order (representing data dependencies) into our algorithm.

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 permutations only (see e.g. [1], or [8] for a discussion). There is related work from other areas, though.

Multidimensional scaling ([3]) 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 almost corresponds 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 ([5]) in 3-dimensional space. Differences result mainly from that we require the algorithm to spread the T values evenly, whereas [5] requires them to be discriminable.

Our algorithm was inspired by Fuzzy Clustering ([2]). 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. 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.

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 course of the optimization process.

Increasing B in the iterative process to first coarsely and later finely search for an optimum is similar in spirit to simulated annealing ([11]) and deterministic annealing ([10]). Finally, our problem is related to the Minimum Bandwidth problem ([4]), except that we are considering weighted graphs and have a different objective function.

Literatur

[1] D.F. Bacon, S.L. Graham, O.J. Sharp, "Compiler Transformations for High-Performance Computing", *ACM Computing Surveys*, 26(4), pp. 345-420, 1994

[2] J.C. Bezdek, "Pattern Recognition with Fuzzy Objective Function Algorithms", Plenum, New York, 1981

[3] T.F. Cox, M.A.A. Cox, "Multidimensional Scaling", Chapman & Hall, 1994

[4] P. Crescenzi, V. Kann, "A compendium of NP optimization problems", 1995, ftp nada.kth.se: Theory/Viggo-Kann/compendium.ps.Z, GT37

[5] K.M. Fairchild, S.E. Poltrock, G.W. Furnas: "SemNet: Three-Dimensional Graphic Representations of Large Knowledge Bases", Lawrence Erlbaum Associates, 1988

[6] B.W. Kernighan, S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Systems Tech. J.* 49 (1970), pp. 291-307

[7] E.L. Lawler (ed.), "The Traveling salesman problem", Wiley, 1992

[8] C. Leopold, "A Fuzzy Approach to Automatic Data Locality Optimization", *Proc. ACM Symp. on Applied Computing*, pp. 515-518, 1996

[9] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, "Numerical Recipes", Cambridge University Press, 1986

[10] K. Rose, E. Gurewitz, G.C. Fox, "Constrained Clustering as an Optimization Method", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 8, pp. 785-794, 1993

[11] S.C. Shapiro, "Encyclopedia of artificial intelligence", Wiley, 1992