アルゴリズム 75－7
（2000．11．10）

平面巡回セールスマン問題に対するアローラの近似アルゴリズムの効率的実装

高橋一寿　　玉木 久夫
明治大学大学院理工学研究科
{kazu35, tamaki}@cs.meiji.ac.jp

概要

平面巡回セールスマン問題に対するアローラの動的計画法近似アルゴリズムの効率的な実装を行ったので報告する。効率化は、動的計画法に現れる部分問題をその分解法のリストへ写像するためにいくつかのあらかじめ計算された表を用いることにより得られる。この実装を用いてTSPLIBにある幾何的例題ほぼすべてに対して実験を行った。

An efficient implementation of the Arora's approximation algorithm for the traveling salesman problem in the plane

Kazutoshi Takahashi　　Hisao Tamaki
School of Science and Technology, Meiji University

Abstract

We give an efficient implementation of the Arora's dynamic programming algorithm for approximating the Euclidean traveling salesman problem in the plane. The efficiency results from certain precomputed tables that are used to map a particular subproblem occurring in the dynamic program to the list of its decompositions into further subproblems. This implementation, combined with some heuristics for the plane decomposition and portal selection, is experimented on most geometric instances of TSPLIB.

1 Introduction

Given a finite set S of points called cities in the plane, the Euclidean traveling salesman problem (Euclidean TSP) asks for a tour of the minimum Euclidean length that visits all the points of S. The Euclidean TSP, as well as the general TSP, in which the distance between each pair of cities are specified in the problem instance, has been studied by many researchers. See Lawler et. al. [6]for early developments on general and Euclidean TSP and Johnson and McGeoch [5] for more recent work, especially in the framework of local search heuristics.

The Euclidean TSP is NP-hard(see [4] for references) and the question of whether it admits a polynomial time approximation scheme had received a considerable amount of attention, until Arora[1] answered the question in the affirmative. That is, for each positive constant ϵ, he constructs a polynomial time algorithm that for every input computes a solution of length within $(1+\epsilon)$ times the optimal length. He later improved the result in to a nearly linear time randomized algorithm [2, 3].

Although his result was a theoretical breakthrough, its impact on the practice of solving real instances of Euclidean TSP remains unclear, since there has been no report of serious implementations of his algorithm, to the best of the authors' knowledge. There seem to be good reasons for skepticism: even though the theoretical complexity is described as nearly linear time, the actual upper bound is $n \log^{c_\epsilon} n$, where c_ϵ is a constant that depends on ϵ and its value that can be extracted from the Arora's analysis is quite large even for a modest value of ϵ. Some researchers (including ourselves, in the early stage of our work) seem to be discouraged by the performance of a casual implementation that spends hours on a small instance (say with several hundred cities) and produces a solution of miserable quality (say more than 20 percent longer than the optimal).

In this report, we describe an efficient implementation of Arora's algorithm that is at least worth experimenting on. It easily handles the geometric instances in TSPLIB with up to 30,000 cities. We describe the results of preliminary experiments.

2 Arora's dynamic programming scheme

Arora's polynomial time approximation scheme uses dynamic programming that is based on a recursive decomposition of the bounding rectangle of the entire city set into smaller rectangles. Given a set S of cities on the plane, a *decomposition tree* for S is a rooted binary tree D that has the following structure. To each node v of D, a rectangle $R_D(v)$ is associated. For the root r of D, $R_D(r)$ is the bounding rectangle of S. If v is a non-leaf node of D and v_1 and v_2 are two child nodes of v, then $R_D(v_1)$ and $R_D(v_2)$ are the two rectangles formed from $R_D(v)$ by cutting it by a line segment parallel to one of its sides. This line segment is called the *divider* of $R_D(v)$ in D and denoted by $\partial_D(v)$. We assume that no city in S is on any divider. The *granularity* of D is the maximum number of cities contained in a leaf rectangle of D: $\max_v |R_D(v) \cap S|$, where the maximum is taken over all the leaf nodes of D.

Let D be a decomposition tree. A *portaling* P on D assigns each non-leaf node v of D a finite set $P(v)$ of points on $\partial_D(v)$. Each point in $P(v)$ is called a *portal*. We denote by $\hat{P}$ the set of all portals $\bigcup_v P(v)$, where the union is over all the non-leaf nodes of D. We define the *degree* of P to be the maximum number of portals on a divider: $\max_v |P(v)|$, where the maximum is taken over all the non-leaf nodes of D.

Let S be a set of cities, D a decomposition tree for S and P a portaling on D. An (S, D, P)-*tour* is a cycle on $S \cup Q$, where $Q \subseteq \hat{P}$, in which each edge, drawn as a straight line segment, crosses no divider of D. Thus, an (S, D, P)-tour is a salesman tour "detoured" by going through portals when crossing dividers of D. Let

$$\epsilon(S, D, P) = \frac{OPT(S, D, P) - OPT(S)}{OPT(S)}$$

where $OPT(S)$ is the length of the optimal (shortest) salesman tour on oS and $OPT(S, D, P)$ is the length of the optimal (S, D, P)-tour. This quantity represents the relative error within which the pair (D, P) of decomposition/portaling captures the optimal salesman tour.

Let $|S| = n$ and let ϵ be an arbitrary positive constant that does not depend on n. Arora[1] showed the existence of a decomposition tree D for S of granularity 1 and a portaling P of degree $O(\log n)$ such that $\epsilon(S, D, P) \leq \epsilon$. He constructed a dynamic programming algorithm that finds such D and P together with the optimal (S, D, P)-tour in polynomial time. He later extended the result by giving an efficient randomized scheme to provide a pair (D, P) with the same property as above with probability at least half[1] [2, 3].

In most of this paper, we assume that a decomposition D and a portaling P with small $\epsilon(S, D, P)$ are given and focus on how to compute the optimal (S, D, P)-tour. We later describe some heuristics for obtaining a plausible decomposition and portaling. In the rest of this section, we describe and analyze the dynamic programming scheme of Arora. We fix a city set S, a decomposition tree D for S and a portaling P on D.

For each node v of D, let $S(v) = S \cap R_D(v)$ denote the set of cities inside the rectangle $R_D(v)$ and let $\tilde{P}(v)$ denote the set of all portals on the perimeter of $R_D(v)$. Note that each side of $R_D(v)$ is a part (or the whole) of the divider of an ancestor node of v. A *portal-pairing* on v is a graph of maximum degree 1 on the vertex set $\tilde{P}(v)$. A portal pairing of v is said to be *non-crossing* if it can be drawn without crossing edges when each vertex in $\tilde{P}(v)$ is drawn on its position on the perimeter of $R_D(v)$ as a portal and each edge is drawn as a curve inside $R_D(v)$ connecting the endvertices.

Let v be a non-root node of D. A *path cover* on v is an acyclic graph on $S(v) \cup \tilde{P}(v)$ in which the degree of each city is exactly two and the degree of each portal is either 0 or 1. In other words, a path cover consists of paths endpointed by portals and collectively going through all the cities inside the rectangle, possibly with isolated portals. The *length* of a path cover is defined to be the sum of the lengths of the edges in the path cover. We say that a path cover C on v *conforms to* a portal pairing π on v if there is a one-to-one correspondence f from the edges (pairs) of π to the paths of C such that for each pair $\{p, q\}$ of portals in π the endpoints of the path $f(\{p, q\})$ are p and q.

For each non-root node v of D and each non-crossing portal pairing π on v, let best(v, π) denote the length of the shortest path cover on v conforming to π. Arora's dynamic programming consists in constructing a table that stores best(v, π) for each v and π. Suppose this task has been done. Then, the $OPT(S, D, P)$ is computed as follows. Let v_1 and v_2 be the two child nodes of the root of D.

[1] His theorem, that leads to the nearly linear time randomized approximation scheme, is stronger than stated above, but we do not need the full strength here.

$$OPT(S, D, P) = \min_{\pi_1, \pi_2} \text{best}(v_1, \pi_1) + \text{best}(v_2, \pi_2)$$

where the minimum is taken over all pairs (π_1, π_2) where π_i is a portal pairing on v_i $(i = 1, 2)$ such that the union of the edges in π_1 and π_2 form a single connected cycle.

We now describe how the dynamic programming table (DP table) is constructed. The table of $\text{best}(v, \pi)$ for each leaf node v is computed in any brute force manner. Assuming that the granularity of D is a constant c, the cost of computing $\text{best}(v, \pi)$ is $O(|\pi|^c)$ where $|\pi|$ denotes the number of edges in π. Let us turn to a non-leaf, non-root node v and assume the table entries for its two child nodes v_1 and v_2 have already been computed. Let π, π_1, and π_2 be portal pairings on v, v_1, and v_2 respectively. We say that π_1 and π_2 *composes* π if the graph consisting of the edges of π_1 and π_2 is acyclic and consists of a set of paths such that the set of endpoint pairs of these paths equals the set of pairs (edges) of π. Then, for each portal pairing π on v, $\text{best}(v, \pi)$ is computed by

$$\text{best}(v, \pi) = \min_{\pi_1, \pi_2} \text{best}(v_1, \pi_1) + \text{best}(v_2, \pi_2)$$

where the minimum is taken over all pairs (π_1, π_2) that composes π.

It should be clear that all the table entries can be computed in a bottom-up manner using the above recurrence. Once $OPT(S, D, P)$ is computed, it is not difficult to recursively identify the portal pairing at each node that is used for this optimal solution and construct an optimal tour.

The number of non-crossing portal pairing at node v is exponential in $|\tilde{P}(v)|$, the number of portals around the rectangle $R_D(v)$. This is acceptable if our purpose is only to show a polynomial time bound since $|\tilde{P}(v)|$ can be bounded by $O(\log n)$. We need a closer look, however, if we are interested in actually running this algorithm.

Let $M = |\tilde{P}(v)|$ be the number of portals around the boundary of rectangle $R_D(v)$. Then the number of non-crossing portal pairings at v is $K(M) = \sum_{0 \le k \le M/2} \binom{M}{2k} C_k$, where $C_k = \binom{2k}{k}/(k+1)$ is the kth Catalan number (see [3] but note the different formulations). To list a few actual values, $K(10) = 2188$, $K(11) = 5798$, $K(12) = 15511$, $K(13) = 41835$, $K(14) = 113635$, and $K(15) = 310572$. Since the task of computing the optimal value for each subproblem specified by a single portal pairing is computationally nontrivial, it is probably not feasible in practical sense to have more than 13 or 14 portals around a rectangle. Two questions arise:

1. Can the dynamic programming be run in a reasonable amount of time, given a portaling that places up to 14 portals around a rectangle?
2. Is the limit of 14 portals around a rectangle sufficient for obtaining reasonably close approximate solutions?

The first question depends on how fast we can process one entry in the DP table. The table size of 113634 may not be formidable itself. In the next section we describe implementation techniques that enable an affirmative answer to this question.

The second question depends on the decomposition and portaling we use. Experiments show that a decomposition and portaling that are insensitive to the given set of cities tend to produce rather miserable approximations, while simple heuristics for better decomposition and portaling considerably improve the performance. The results of the experiments we made are described in Section4.

3 Implementation of the dynamic programming

The key to an efficient implementation of the dynamic programming procedure described in the previous section is a compact encoding of the portal pairings.

Fix a decomposition tree D and portaling P and let π be a portal pairing on some node v of D. Let P_π denote the set of portals that are paired by π, or, more formally, the set of vertices of degree 1 in the graph π. We call P_π the *support* of π. Clearly, the support of any portal pairing must consist of even number of portals. A crucial observation, that appears already in Arora's analysis[1, 3], is that given a support Q of cardinality $2n$, each non-crossing pairing with support Q can be identified with a

well-formed sequence of $2n$ parentheses. Here, "well-formed" means that there are equal numbers of left and right parentheses and, moreover, in any prefix of the sequence the number of right parentheses does not exceed the number of left parentheses. Given a well-formed sequence ω of $2n$ parentheses, the pairing π_ω with support Q that corresponds to ω is naturally determined as follows. Let the portals in Q be listed in the clockwise order as $\{p_1, p_2, \ldots, p_{2n}$. Then, for any pair $i < j$, p_i and p_j are paired in π_ω if and only if the ith and the jth symbols in ω are left and right parentheses that "match" each other in the usual sense.

Thus, our representation of a portal pairing at node v consists of two bit vectors: the *selection vector* u of length $|\tilde{P}(v)|$ that specifies the support and the *connection vector* w of length $weight(u)$, that is a well-formed sequence of parentheses, with left and right parentheses represented by 0 and 1 respectively. Here, $weight(u)$ denotes the number of 1's in the bit vector u.

The compactness of this representation allows us an extensive use of precomputed tables in mapping a subproblem in the dynamic programming computation into a list of decomposing subproblems. Both the selection vector and the connection vector are used as indices of such tables. We use the following tables.

The list of all connection vectors of length $2n$ are listed in table CV_n: for $1 \leq k \leq C_n$, $CV_n[k]$ is the kth connection vector of length n. Recall here that C_n is the nth Catalan number. Though the order can be arbitrary, we adopt the natural order as integers. We also use the inverse map $InvCV_n$: for $0 \leq w < 2^{2n}$, $InvCV[w]$ is k such that $CV_n[k] = w$ if such w is a valid connection vector and -1 otherwise. For each integer $M \geq 0$, all possible pairings for a set of M portals are listed in table PG_M: $PG_M[s].selection$ and $PG_M[s].connection$ are the selection and connection vectors of the sth pairing, for $1 \leq s \leq K(M)$, where $K(M)$ is defined in the previous section. We refer to the index s of table PG_M as the *serial number* of the portal pairing represented by $PG_M[s]$. Given a selection vector u of length M and a connection vector w of length $weight(u)$, the serial number of the portal pairing is determined as a function of u and w. Table SN_M represents this function: $SN_M[u][k]$ is the serial number of the portal pairing with selection vector u and connection vector $CV_{weight(u)}[k]$.

Let π be a portal pairing at node v with children nodes v_1 and v_2. We need an efficient way of listing all pairs (π_1, π_2), where π_i is a portal pairing at v_i, $i = 1, 2$, such that π_1 and π_2 composes π. For this purpose, we prepare one more precomputed table. Fix a pair (π_1, π_2) that composes π and let Q_1 and Q_2 be the supports of π_1 and π_2 respectively. Let $|Q_1| = n_1$, $|Q_2| = n_2$, and $|Q_1 \cap Q_2| = m$. We assume that the portals in Q_1 are listed in the counterclockwise order as $q_1, q_2, \ldots, q_{n_1}$, the portals in Q_2 in the clockwise order as $r_1, r_2, \ldots, r_{n_2}$, with $q_i = r_i$ for $1 \leq i \leq m$. Let $p_1, p_2, \ldots, p_n$, $n = n_1 + n_2 - 2l$, be the clockwise listing of the portals in the support $Q = (Q_1 \cup Q_2) \setminus (Q_1 \cap Q_2)$ of π. Under this convention, the connection vector w_π of π is determined as a function of the connection vectors w_{π_1} and w_{π_2} of π_1 and π_2, respectively, and the number of shared portals r: $w_\pi = f(w_{\pi_1}, w_{pi_2})$. Let m_{max} be the maximum number of portals allowed on a single divider in the decomposition tree. For each k, $1 \leq k \leq C_n$ and r, $1 \leq l \leq m_{max}$, the table entry $CMP_n[k][l]$ of our last table CMP_n stores the list of all pairs (k_1, k_2) such that $f(w_1, w_2, l)$ equals the kth connection vector $CV_n[k]$, where $w_i = CV_{n_i}[k_i]$ is the k_ith connection vector of length n_i, $i = 1, 2$.

Using these tables, one iterative step of the dynamic programming computation at node v with children v_1 and v_2 proceed as follows. The DP table at v is simply an array $best_v$ of length $K(M)$, where M is the number of portals around the rectangle $R_D(v)$. The entry $best_v[s]$ of this table is supposed to be the length of the optimal path cover that conforms to the portal pairing of serial number s. When v is about to be processed, this goal has already been achieved for its children nodes: the tables $best_{v_1}$ and $best_{v_2}$ have been properly established. The following procedure computes $best_v[s]$ for one value of s. We assume a convention in the numbering of portals around a rectangle that is similar to the convention we adopted above for the support sets: the portals around v_1 are numbered counterclockwise, the portals around v_2 are numbered clockwise, with each portal on the divider between v_1 and v_2 given the same number running from 1 to m, where m is the number of such shared portals. The numbering of portals around v is also clockwise and a map map_i, $i = 1, 2$, is provided that maps the number of a portal given at node v to its number given at v_i.

1. Initialize $best_v[s]$ to infinity.
2. Let $u = PG_M[s].selection$ be the selection vector of the portal pairing of serial number s at node v. Based on u and using maps map_1 and map_2, create a partial selection vectors u_1 for v_1 and u_2 for v_2: the $map_i[j]$th bit of u_i is on if and only if the jth bit of u is on, for $i = 1, 2$. The bits corresponding to the m shared portals are tentatively off in u_1 and u_2.

3. For each bit vector t of length m, such that $weight(u_1) + weight(t)$ is even, repeat the following.
(a) Let $n = weight(u)$, $n_1 = weight(u_1) + weight(t)$, $n_2 = weight(u_2) + weight(t)$, and $l = weight(t)$.
(b) For each pair $(k1, k2)$ in the list $CMP_n[k][l]$, repeat the following:
i. Let $s_1 = SN_{n_1}[u_1][k_1]$ and $s_2 = SN_{n_2}[u_2][k_2]$.
ii. If $best_v[s] > best_{v_1}[s_1] + best_{v_2}[s_2]$ then update $best_v[s]$ to $best_{v_1}[s_1] + best_{v_2}[s_2]$.

Owing to the precomputed tables, the innermost loop in the above algorithm is quite compact, involving only several table references per step. Experiments show (see Section 4) that this implementation can handle TSP instances with more than 10,000 cities given a portaling that has up to 14 portals around some (though not all) rectangles.

4 Experiments

We experimented on our implementation in combination with the following decomposition/portaling schemes.

Non-adaptive, randomized decomposition This scheme has two parameters, g, the granularity and m, the number of portals on a single divider. When the number of cities in a rectangle R is greater than g, R is divided into two subrectangles. The divider segment is chosen at random so that it is parallel to the shorter side of R and the ratio between the sizes of two subrectangels is between 1/2 and 2. The number of portals m' on the divider is m when each subrectangle contains more than g cities and thus is subject to further decomposition, and $\lfloor m/2 \rfloor$ if at least one subrectangle is a leaf. On the divider, m' portals are placed so that they partition the divider into $m' + 1$ intervals of equal length.

Heuristic decomposition/portaling This scheme has three parameters g, M and M_{leaf}: g is the granularity as before, M is the maximum number of portals around a single rectangle, and M_{leaf} is the maximum number of portals around a single leaf rectangle. The selection of the divider segment and the placement of portals are based on the following heuristics.

We use a heuristic criterion for an edge between cities to be *eligible*. An eligible edge is a plausible candidate for a TSP edge, as considered by the heuristic. The criterion currently adopted is as follows. An edge $\{p, q\}$ is *horizontaly* eligible if the fan of formed by the circle centered at p and passing through q, the segment pq, and the horizontal half line from p towards the direction of q is empty of other cities and vice versa. Vertical eligibility is defined similarly. We do not deny the possiblity that other similar definitions may work better.

Let R be a rectangle, d a potential divider of R. Assume d is vertical, and let C be the set of points on d at which horizontaly eligible edges cross d. For a set Q of points on d, we define the penalty of Q, penalty(Q) to be $\min_\alpha \max_p \{\text{dist}(p, \alpha(p)) \mid p \in C\}$ where $\alpha : C \to Q$ ranges over all the assignments of the points of C to the points in Q. The intention of penalty(Q) is to measure the loss incurred when the TSP tour using eligible edges is detoured by going through portals place at points in Q. For positive integer k, we define penalty(d, k) to be $\min_Q \{\text{penalty}(Q) \mid Q \text{ a set of } k \text{ points on } d\}$. Thus, penalty$(d, k)$ is a rather pessimistic evaluation of the badness of the divider d, when k portals are allowed to be placed. Note that penalty(d, k) can be efficiently computed via dynamic programming. Our divider selection heuristics are primarily based on the value of penalty(d, k), and use other secondary measures such as sparseness of cities around the divider. Note that the number of portals allowed, as enforced by the parameters M and M_{leaf}, depends on the location of the divider, so this factor must also be considered.

Once a divider is selected and the number k of portals on the divider is determined, the set of portals Q is selected so that it minimizes penalty(Q).

Heuristic decomposition, random portaling This scheme has the same parameters as the previous one. The decomposition is also chosen in the same manner. The selection of the portals, however, are randomized. The idea is that, with the rather tight restriction on the number of portals, the quality of the portal set that is guaranteed by the the optimality of penalty(Q) is rather poor, so we may need to rely on some luck. The hope is that we may hit some good portaling if we try some number of portalings through random choices.

We conducted experiments on most of the geometric instances in TSPLIB: lin318, pcb442, disj1000, pr1002, pr2392, pcb3038, fl3795, fnl4461, pla7397, usa13509, pla338101, and pla85900. The number in the instance name represents the number of cities in the instance. In the tables listed, instances are referred to by these numbers only. The computer used is SUN Ultra60 with UltraSparcII, 360MHz, and

1GB memory.

Tables 5 and 2 summarize the results for the non-adaptive decomposition/portaling, with two sets of parameters. Note that, in the second table where $m = 5$, the number of portals around some rectangle reaches 14. The quality of the solution is measured by the excess of the solution, i.e., its length minus the optimal, in terms of percents. The row "DP excess" lists the excess of the raw dynamic programming solution and the row "Short Cut" lists the excess of the solution obtained from short-cutting portals. On these short-cut tours, a simple postprocessing optimization, which we call Frame-Opt, is performed and the results of this are listed as well. Since the non-adaptive decomposition is randomized, we made 10 trials for each instance and listed average, minimum and maximum values for each table entry.

Tables 3 and 4 summarize the results for the heuristic decomposition. The number of portals allowed around a single rectangle is 10 and 12 respectively. In most instances, significant improvements are observed over the non-adaptive method, in all terms of the table size, running time, and quality of the solution.

Table 5 is for the heuristic decomposition with random portaling. This strategy sometimes outperforms the deterministic portaling, but its average performance is not as good as the deterministic strategy, as expected.

The last table, Table 6 lists result for the larger instances of the TSPLIB.

In three cases, our implementation reported infeasibility of the problem (fnl3795 in Table 3 and pla85900 in Table 6). Although this is quite likely be a programming error, there is some possibility that those (S, D, P) problem instances are indeed infeasible. Despite our extensive effort, no bug has been identified. Note that the certificate of infeasibility can be almost as large as the total DP table and it is not an easy task to distinguish between a sophisticated bug and real infeasibility.

5 Concluding remarks

Our implementation has made it possible to experiment the Arora's algorithm on quite large instances. Its performance is still poor compared to the best-known heuristics such as Lin-Kernighan[7](see also [5] for a recent demonstration of its remarkable performance). We are currently experimenting on more refined heuristics for decomposition/portaling in hope of getting better results. We are also considering the possibility of performing local search in the space of portalings. If the decomposition tree is fixed and once the dynamic programming tables are completed for one portaling, relatively little computation is required to update the table when a small change is made to the portaling. If the infeasibility observed in the experiment turns out real, a serious attension must be paid to this issue.

参考文献

[1] S. Arora, "Polynomial time approximation schemes for Euclidean TSP and other geometric problems," Proc. 37th Annual IEEE Symposium on Foundation of Computer Sciences, 2-11, 1996.

[2] S. Arora, "Nearly liner time approximation schemes for Euclidean TSP and other geometric problems," Proc. 38th Annual IEEE Symposium on Foundation of Computer Sciences, 554-563, 1997.

[3] S. Arora, "Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems," Journal of the ACM 45(5), 753-782, 1998.

[4] M. R. Garey, D. S. Johnson, Computers and Intractability, Freeman, New York, 1997.

[5] D. S. Johnson, L. A. Mcgeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," in Local Search in Combinatorial Optimization, E.H.L. Aarts and J.K. Lenstra (eds.), John Wiley and Sons, NY, 1997.

[6] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys, eds., "The traveling salesman problem," J. Wiley & Sons, 1985.

[7] S.Lin, B.W.Kernighan, "An effecive heuristic algorithm for the traveling-salesman problem," Operations Research 21, 498-516, 1973.

instance	318	442	1000	1002	2392	3038	3795	4461	7397
table size (k)	32	54	129	156	370	494	427	727	1080
min	25	35	95	100	308	444	356	672	960
max	42	88	169	218	406	533	502	806	1144
DP excess (%)avg	19.0	20.8	23.7	22.8	26.0	20.6	43.2	21.2	23.1
min	14.7	17.5	21.7	21.5	25.3	19.6	38.3	20.0	21.4
max	26.4	23.1	26.3	24.1	27.5	22.2	50.2	22.3	24.4
Short Cut (%) avg	9.66	11.2	14.1	13.6	15.0	12.4	18.5	13.3	12.4
min	5.58	7.97	13.1	12.5	13.7	11.5	15.1	12.3	11.1
max	16.4	14.1	16.4	15.1	16.3	14.1	22.5	14.0	13.2
time DP(s) avg	2.6	5.8	12.1	15.8	37.4	42.6	32.0	60.4	140.6
min	1.6	3.48	7.7	11.7	31.9	34.6	27.2	53.9	117.5
max	4.1	9.8	22.5	26.8	45.9	58.2	40.9	71.7	198.2
Frame-Opt(%)avg	3.19	3.81	4.20	4.00	4.75	4.62	10.5	4.00	5.15
min	1.53	2.47	3.41	3.36	4.25	3.90	6.66	3.61	4.55
max	4.46	5.77	4.86	4.67	5.58	5.06	13.5	4.52	5.62
time F-Opt(s)avg	2.3	4.0	13.7	8.4	17.5	29.4	60.5	43.0	185.9
min	1.5	2.2	11.3	5.1	15.6	27.3	47.0	38.9	94.6
max	3.9	5.1	19.7	10.1	22.4	39.4	90.1	54.0	472.5

表 1: Non-adaptive decomposition/portaling, $g = 6, m = 4$

instance	318	442	1000	1002	2392	3038	3795	4461	7397
table size (k)	243	373	987	1040	2657	3650	3619	5396	8032
min	136	291	767	754	2439	3320	3001	4787	7577
max	324	456	1181	1278	2992	4144	3986	6006	8817
DP excess (%)avg	19.1	19.6	22.7	22.4	26.1	20.8	39.0	20.6	23.1
min	13.7	17.0	20.9	21.0	24.9	19.5	35.7	20.0	22.0
max	24.2	22.1	24.8	24.3	27.2	22.2	41.9	21.1	24.5
Short Cut(%) avg	8.77	8.51	11.9	11.9	13.5	11.1	15.4	11.8	10.7
min	5.76	5.81	10.8	10.7	12.1	10.0	12.5	11.4	9.59
max	12.1	10.1	13.3	13.7	14.7	12.6	18.8	12.1	12.0
time DP(s) avg	10.9	14.1	54.2	54.5	135.1	182.9	164.5	270.2	426.9
min	4.6	10.2	40.8	31.9	117.6	162.1	138.5	224.8	366.8
max	16.1	23.8	66.0	92.1	151.8	211.2	182.3	301.5	507.1
Frame-Opt(%)avg	2.55	3.08	3.99	3.39	4.57	4.22	7.91	3.76	4.53
min	1.56	1.80	3.19	2.89	3.67	3.64	6.31	3.58	4.02
max	3.39	5.02	4.69	4.37	5.33	4.98	10.3	3.96	5.22
time F-Opt(s)avg	2.3	3.2	10.7	7.9	17.1	28.7	64.6	37.4	223.4
min	1.0	2.1	6.6	5.5	15.9	25.1	41.2	35.8	83.9
max	3.4	4.1	14.4	9.5	18.8	35.9	87.1	39.6	432.5

表 2: Non-adaptive decomposition/portaling, $g = 4, m = 5$

instance	318	442	1000	1002	2392	3038	3795	4461	7397
time Decomp (s)	0.2	0.3	1.4	1.3	2.5	3.8	3.0	5.6	8.0
table size (k)	194	229	430	443	916	1172	1445	1482	2043
DP excess(%)	9.80	11.93	12.53	11.33	12.25	11.96	∞	13.89	11.78
Short Cut(%)	3.54	3.05	5.95	4.71	6.07	6.06	∞	8.40	5.22
time DP(s)	5.1	5.3	10.1	11.3	22.2	27.4	31.9	40.9	58.4
Frame-Opt(%)	2.19	1.36	3.02	2.37	2.39	2.40		3.44	2.84
time F-Opt(s)	1.7	2.3	5.7	4.6	9.8	16.7		38.0	50.0

表 3: Heuristic decompostion/portaling, $g = 6$, $M = 10$, $M_{leaf} = 6$

instance	318	442	1000	1002	2392	3038	3795	4461	7397
time Decomp (s)	0.2	0.3	1.4	1.3	2.4	3.6	2.6	5.3	7.3
table size (k)	599	790	953	941	1323	1576	2575	1993	3075
DP excess(%)	4.89	6.35	6.71	6.98	7.99	7.94	12.69	9.68	7.56
Short Cut(%)	1.15	2.57	3.64	4.05	4.84	4.83	8.00	6.75	3.88
time DP(s)	30.7	46.5	55.2	59.3	94.9	116.5	110.8	150.0	210.6
Frame-Opt(%)	0.63	0.97	1.46	1.65	2.46	2.49	4.97	3.19	2.35
time F-Opt(s)	1.2	2.5	4.9	5.2	10.0	29.7	35.7	37.2	157.8

表 4: Heuristic decomposition/portaling, $g = 6$, $M = 12$, $M_{leaf} = 8$

instance	318	442	1000	1002	2392	3038	3795	4461	7397
time Decomp (s)	0.2	0.3	1.4	1.3	2.4	3.6	2.6	5.3	7.3
table size (k) avg	606	745	955	934	1318	1571	2630	1951	3177
min	587	712	908	883	1295	1546	2556	1934	3120
max	629	786	992	1008	1357	1605	2719	2010	3227
DP excess (%)avg	5.87	7.29	8.26	7.27	7.48	7.83	14.9	9.57	7.98
min	5.22	6.24	7.46	6.90	7.12	7.61	12.1	9.46	7.72
max	7.81	7.84	9.38	7.82	7.86	7.99	17.0	9.73	8.28
Short Cut(%) avg	1.71	3.24	4.98	4.20	4.53	4.63	9.20	6.57	4.35
min	0.85	2.46	4.34	3.71	4.15	4.48	7.29	6.37	3.98
max	2.75	3.91	6.21	4.65	4.94	4.77	11.0	6.75	4.66
time DP(s) avg	32.56	40.3	52.6	60.8	95.1	110.5	117.6	151.6	220.8
min	30.90	37.7	46.8	54.3	92.8	106.6	108.7	144.8	215.4
max	34.46	43.6	55.5	70.6	100.6	119.3	135.3	171.3	228.6
Frame-Opt(%)avg	0.90	1.43	2.44	2.02	2.23	2.23	4.83	3.08	2.67
min	0.26	0.53	2.05	1.55	1.88	2.00	3.90	2.93	2.26
max	2.15	2.21	2.80	2.70	2.58	2.49	6.22	3.23	2.96
time F-Opt(s)avg	1.45	2.7	7.2	5.5	11.1	17.3	40.9	33.5	124.0
min	1.06	2.2	5.2	4.6	9.3	16.2	36.9	23.2	50.7
max	2.17	3.7	9.4	8.0	15.5	18.2	44.7	41.9	189.5

表 5: Heuristic decomposition, random portaling, $g = 6$, $M = 12$, $M_{leaf} = 8$

parameter	$g = 6$, $M = 10$, $M_{leaf} = 6$				$g = 6$, $M = 12$, $M_{leaf} = 8$			
instance	13509	18512	33810	85900	13509	18512	33810	85900
time Decomp (s)	17.9	35.2	83.2	413.1	16.4	32.0	70.4	339.2
table size (k)	2460	3716	6991	13876	3055	4887	9555	17567
DP excess(%)	18.43	15.41	20.62	∞	15.30	12.63	16.69	∞
Short Cut(%)	11.57	9.53	13.46	∞	10.27	8.47	12.19	∞
time DP(s)	80.7	113.8	242.7	607.7	332.9	461.7	798.6	1293.9
Frame-Opt(%)	4.28	3.60	4.79		4.05	3.59	4.40	
time F-Opt(s)	235.9	305.7	936.3		269.9	351.2	875.4	

表 6: Larger instances: heuristic decomposition/portaling