Missouri Estimation of Distribution Algorithms Laboratory

Dependency Trees, Permutations, and Quadratic Assignment Problem

Martin Pelikan, Shigeyoshi Tsutsui, and Rajiv Kalapala

MEDAL Report No. 2007003

January 2007

Abstract

This paper describes and analyzes an estimation of distribution algorithm based on dependency tree models (dtEDA), which can explicitly encode probabilistic models for permutations. dtEDA is tested on deceptive ordering problems and a number of instances of the quadratic assignment problem. The performance of dtEDA is compared to that of the standard genetic algorithm with the partially matched crossover (PMX) and the linear order crossover (LOX). In the quadratic assignment problem, the robust tabu search is also included in the comparison.

Keywords

Estimation of distribution algorithm, dependency tree, probabilistic models, implementation, optimization, evolutionary computation,C++.

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis
One University Blvd., St. Louis, MO 63121
E-mail: medal@cs.umsl.edu
WWW: http://medal.cs.umsl.edu/

Dependency Trees, Permutations, and Quadratic Assignment Problem

Martin Pelikan
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 320 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
pelikan@cs.umsl.edu

Shigeyoshi Tsutsui
Dept. of Management and Information Science
Hannan University
5-4-33 Amamihigashi, Matsubara, Osaka 580-5802, Japan
tsutsui@hannan-u.ac.jp

Rajiv Kalapala
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 321 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
rkdnc@cs.umsl.edu

Abstract

This paper describes and analyzes an estimation of distribution algorithm based on dependency tree models (dtEDA), which can explicitly encode probabilistic models for permutations. dtEDA is tested on deceptive ordering problems and a number of instances of the quadratic assignment problem. The performance of dtEDA is compared to that of the standard genetic algorithm with the partially matched crossover (PMX) and the linear order crossover (LOX). In the quadratic assignment problem, the robust tabu search is also included in the comparison.

Keywords: Estimation of distribution algorithm, dependency tree, permutation domain, quadratic assignment problem, linkage learning.

1 Introduction

Estimation of distribution algorithms (EDAs) (Baluja, 1994; Mühlenbein & Paaß, 1996; Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002)—also called probabilistic model-building genetic algorithms (PMBGAs) (Pelikan, Goldberg, & Lobo, 2002; Pelikan, 2005) and iterated density estimation algorithms (IDEAs) (Bosman & Thierens, 2000)—replace standard crossover and mutation operators of genetic and evolutionary algorithms by building a probabilistic model of selected solutions and sampling the built model to generate new candidate solutions. EDAs have been shown to provide solutions to previously intractable problems and outperform other approaches

in broad classes of challenging optimization problems (Larrañaga & Lozano, 2002; Mühlenbein & Mahnig, 1999; Pelikan, 2005; Pelikan, Sastry, & Cantú-Paz, 2006). Since in many important real world problems—such as the quadratic assignment problem, scheduling, and vehicle routing—candidate solutions can be represented by permutations, designing EDAs for solving problems defined over permutations represents an important challenge. Nonetheless, only few studies exist that focus on the design and application of EDAs that directly manipulate solutions encoded as permutations (Bengoetxea, Larrañaga, Bloch, Perchant, & Boeres, 2000; Bosman & Thierens, 2001b; Tsutsui, Goldberg, & Pelikan, 2002; Zhang, Sun, Tsang, & Ford, 2003; Tsutsui, Pelikan, & Goldberg, 2006).

The purpose of this paper is to describe and analyze an EDA based on dependency tree models (dtEDA), which can explicitly encode probabilistic models for permutations, encoding both the absolute positions of different permutation elements as well as their relative ordering constraints. dtEDA for permutations can be easily generalized to other types of probabilistic models, such as multiply connected Bayesian networks of the Bayesian optimization algorithm (BOA) or marginal product models of the extended compact genetic algorithm (ECGA). Nonetheless, due to the complexity of model building for permutations, using multivariate models may not lead to practical performance improvements for typical permutation problems. As test problems, we consider deceptive ordering problems (both the absolute ordering problem and the relative ordering problem) and the quadratic assignment problem (QAP). To compare dtEDA for permutations with other methods, we also include two of the standard two-parent permutation-based recombination operators: the partially matched crossover (PMX) (Goldberg & Lingle, 1985; Goldberg, 1989) and the linear order crossover (LOX) (Falkenauer & Bouffouix, 1991). Additionally, on QAP, we also include robust tabu search (Taillard, 1991), which is one of the standard methods for solving QAP.

The paper is organized as follows. Section 2 introduces some of the past approaches to solving permutation problems with EDAs. Section 3 describes the dependency tree EDA and its extension to permutation problems. Section 4 describes test problems used in the experiments. Section 5 presents and discusses experimental results. Section 6 discusses important topics for future work. Section 7 summarizes and concludes the paper.

2 Solving Permutation Problems with EDAs

Estimation of distribution algorithms (EDAs) (Baluja, 1994; Mühlenbein & Paaß, 1996; Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002) evolve a population (multiset) of potential solutions to the given optimization problem using a combination of techniques from evolutionary computation and machine learning. Like standard evolutionary algorithms (Goldberg, 1989; Holland, 1975), EDAs start with a randomly generated population of potential solutions. Each iteration updates the population using selection and variation operators. The selection operator selects a population of promising solutions from the current population. However, instead of using variation operators inspired by natural evolution and genetics, such as two-parent crossover and mutation, EDAs create new solutions by *building an explicit probabilistic model* of selected solutions and *sampling the built model* to generate new candidate solutions. The new candidate solutions are incorporated into the original population and the next iteration is executed unless some termination criteria have been met.

Ideally, the quality of candidate solutions generated by the probabilistic model improves over time, and, after a reasonable number of iterations, the model should generate only the best optima. The EDA pseudocode can be found Figure 1.

```
Estimation of distribution algorithm (EDA)
  t := 0;
  generate initial population P(0);
  while (not done) {
    select population of promising solutions S(t);
    build probabilistic model M(t) for S(t);
    sample M(t) to generate O(t);
    incorporate O(t) into P(t);
    t := t+1;
  };
```

Figure 1: Pseudocode of the estimation of distribution algorithm (EDA).

2.1 Permutation Problems and Regularities

Let us first identify factors that typically influence solution quality in permutation problems. In some problems, solution quality depends mainly on the relative ordering of pairs of permutation elements and the task is to find a permutation to satisfy a set of relative-ordering constraints; an example problem of this type is job-shop scheduling. Sometimes solution quality depends mainly on the pairs of elements that are located next to each other (neighbors), which is a typical feature of the traveling salesman problem and similar problems. Finally, in some permutation problems solution quality depends also on absolute positions of different permutation elements; an example problem of this form is the quadratic assignment problem.

While sometimes the various constraints or solution features influence solution quality independently, sometimes it is necessary to also consider correlations between the different features. To illustrate how correlations may become an important factor in real-world permutation-based problems, we can consider the quadratic assignment problem, where absolute positions of the various elements directly influence the fitness but in addition to this, absolute positions of some elements significantly influence good absolute positions of some other elements (depending on the distance and flow matrices).

To design effective variation operators for broad classes of permutation-based problems, it is important to ensure that the variation operators are capable of exploiting the above types of regularities and recombining the good features among subsets of candidate solutions effectively. Thus, we would like to use variation operators that are able to discover and preserve (1) relative ordering constraints, (2) absolute ordering constraints, (3) neighbor relations, and (4) correlations between these factors.

Two approaches have been proposed to solving permutation problems with EDAs: (1) Real-valued EDAs with random keys and (2) EDAs with explicit permutation models. The remainder of this section describes these two types of approaches.

2.2 Random Keys and Real-Valued EDAs

Random keys (Bean, 1994) enable evolutionary algorithms to represent permutations by fixed-length real-valued vectors and because many evolutionary algorithms have been designed to work with real-valued vectors, random keys have become a popular approach to solving permutation problems (Bryan, Alice, & Smith, 1997; Knjazew & Goldberg, 2000; Bosman & Thierens, 2001b; Rothlauf, Goldberg, & Heinzl, 2002; Rothlauf, 2006). To solve permutation problems with real-

valued EDAs, we can use practically any real-valued EDA and modify the fitness function so that it first decodes each real-valued vector into a permutation and evaluates the resulting permutation to determine the solution's fitness.

To determine the permutation that is encoded by a real-valued vector $(x_0, x_1, \ldots, x_{n-1})$, all elements of the vector are first (temporarily) sorted. The final position of x_i determines the position of i in the permutation. Any real-coded vector can be thus easily transformed into a permutation although when two or more coordinates in the vector are equal, the permutation is not unique (ties can be resolved arbitrarily as it is very unlikely that such cases occur in practice).

As an example, let us consider the vector $(0.3, 0.7, 0.5, 0.1)$. Sorting this vector leads to $(0.1, 0.3, 0.5, 0.7)$. The first element of the original vector, $x_0 = 0.3$, gets to the second position of the sorted vector and that is why 0 goes to the second position of the resulting permutation. The second element of the original vector, $x_1 = 0.7$, gets to the last position of the sorted vector and that is why 1 is located in the last position of the resulting permutation. Other elements are processed analogically, leading to the permutation $(3, 0, 2, 1)$.

The advantage of using random keys for solving permutation problems is that random keys allow us to use practically any real-valued optimizer to optimize permutation problems. On the other hand, standard variation operators for real-valued representations can be expected to fail to effectively process problem regularities with random keys due to the number and variety of the ways of encoding each relative or absolute ordering constraint. As an example, consider a specific pair of positions in the random-key representation and consider the following pairs of values for these two positions: $(0.1, 0.9)$, $(0.01, 0.2)$, $(0.03, 0.04)$, and $(0.84, 0.9)$. While all these random-key value pairs represent the same relative ordering constraint (the first element precedes the second element), from the perspective of real-valued optimization, the vectors have very little in common. The ineffectiveness and the large overhead introduced by random key representations in real-valued EDAs has been analyzed in Bosman and Thierens (2001a).

Past EDAs for the random-key representation of permutations (Bosman & Thierens, 2001b; Bosman & Thierens, 2001a; Robles, de Miguel, & Larrañaga, 2001) use multivariate normal distributions and factorized multivariate normal distributions to model promising solutions. However, since it has been shown that using multivariate normal distributions for learning and sampling permutations represented as random keys does not lead to good performance (Bosman & Thierens, 2001a), it has been proposed that the probabilistic model be used only as the source of information about the dependencies between the different positions in a permutation and that the obtained linkage information be used in a specialized crossover that directly manipulates the discovered building blocks or linkage groups (Bosman & Thierens, 2001b). The proposed approach was called the IDEA induced chromosome elements exchanger or $\mathbb{ICE}$. $\mathbb{ICE}$ has been shown to significantly outperform standard EDAs that use factorized multivariate distributions to model and sample solutions represented by random keys.

Besides the redundancy of the encoding and the resulting overhead, high multimodality represents yet another potential problem for random key representations and the EDAs that use a single multivariate normal distribution as a model. Since many difficult permutation problems—such as the quadratic assignment problem and the vehicle routing problem—are highly multimodal, using real-valued EDAs may necessitate the use of more complex models than those explored in the past work on using EDAs with random keys.

2.3 EDAs with Explicit Probabilistic Models for Permutations

While it is straightforward to model discrete, fixed-length vectors with multivariate probabilistic models such as Bayesian networks, using standard model-building and model-sampling procedures for permutations will lead to many vectors that are not valid permutations (some of the symbols will be repeated while some will be missing). There are three straightforward approaches to fix this problem:

1. modify model building and sampling procedures to ensure that only valid permutations are sampled,
2. introduce a repair operator that can fix invalid permutations, or
3. reject all invalid permutations and resample.

Most EDAs for permutation problems take the first approach and modify the model sampling procedure in order to sample only valid permutations (Bengoetxea, Larrañaga, Bloch, Perchant, & Boeres, 2000; Tsutsui, Goldberg, & Pelikan, 2002; Zhang, Sun, Tsang, & Ford, 2003; Tsutsui, Pelikan, & Goldberg, 2006) although the remaining two approaches have also been analyzed in graph matching problems (Bengoetxea, Larrañaga, Bloch, Perchant, & Boeres, 2000). Although not formally called EDAs, ant colony approaches to solving permutation problems (Stutzle, 1998; Gambardella, Taillard, & Dorigo, 1999; Merkle & Middendorf, 2005) also base their exploration on the sampling of a probabilistic model and have been successfully applied to solving permutation problems with the sampling restricted to only valid permutations.

In the edge-histogram based sampling algorithm (EHBSA) (Tsutsui, Goldberg, & Pelikan, 2002; Tsutsui, Pelikan, & Goldberg, 2006) and the EDA with guided local search (EDA/GLS) (Zhang, Sun, Tsang, & Ford, 2003), the models primarily focus on encoding relative ordering constraints between neighbors. On the other hand, the node histogram based sampling algorithm (NHBSA) (Tsutsui, Pelikan, & Goldberg, 2006) models each position independently and focuses on preserving absolute positions of permutation elements directly. Additionally, EHBSA and NHBSA use a template to bias the new individuals to the best solution seen so far while EDA/GLS uses the sampling of the probabilistic model to guide mutation of the candidate solutions.

EDAs proposed in Bengoetxea et al. (2000) represent the closest approach to that studied in this paper and the sampling procedure described in this paper is derived from that work. Nonetheless, here the algorithm is applied to deceptive permutation problems and the quadratic assignment problem, whereas Bengoetxea et al. (2000) consider solving inexact graph matching problems. Furthermore, here we consider decision tree models as they represent a practical compromise between univariate models and Bayesian networks. While univariate models oversimplify the problem and allow preservation of only absolute position constraints, Bayesian networks may be difficult to learn given the large alphabet size and considering the population-sizing theory of GAs and EDAs (Pelikan, Sastry, & Goldberg, 2002; Pelikan, 2005).

The following section describes the dependency-tree EDA and the modifications of this algorithm that allow its direct application to permutation problems.

3 Dependency-Tree EDA for Permutation Problems

EDAs based on dependency trees were among the first approaches to EDAs that could encode conditional dependencies between problem variables (Baluja & Davies, 1997; Pelikan & Mühlenbein, 1999). In the dependency-tree EDA, the probabilistic model built for the selected population of

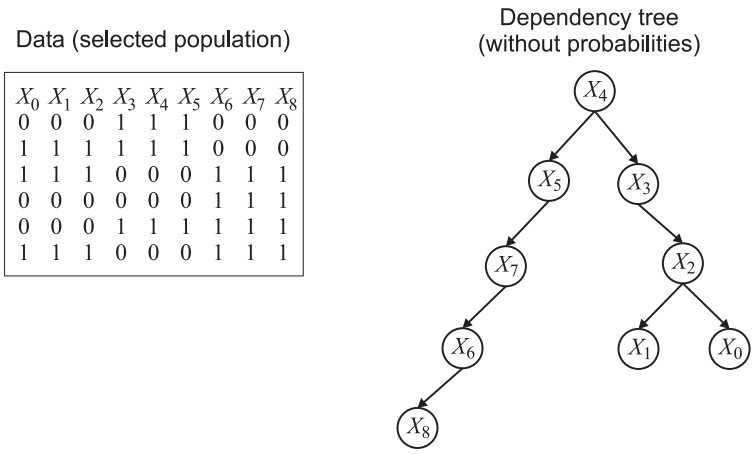

Figure 2: An example dependency tree for a population of 6 strings of 9 bits each. After examining the population, strong correlations can be found among the group of the first three string positions (X_0, X_1, and X_2), and analogical correlations can be found for the groups (X_3, X_4, X_5) and (X_6, X_7, X_8). The dependency tree captures many of these interactions by relating the most strongly correlated variables using conditional dependence. In addition to the structure, the model would still have to define the conditional probabilities of each variable given its parent, which can also be learned from the given sample.

solutions forms a dependency tree (Chow & Liu, 1968; Baluja & Davies, 1997) where each variable is conditioned on its predecessor in the tree. This section first describes standard learning and sampling techniques for dependency trees, and then outlines how these techniques can be modified to permutations.

3.1 Dependency-Tree Models

From EDA perspective, each string position is seen as a random variable and the set of selected solutions is used as a sample from an unknown probability distribution that we want to mimic with our model. A dependency-tree model (Chow & Liu, 1968; Baluja & Davies, 1997) consists of two components: (1) structure, which defines the conditional dependencies between the variables and (2) parameters, which specify marginal and conditional probabilities included in the model.

The structure is a directed tree graph where the nodes correspond to the variables and the edges encode direct conditional dependencies between pairs of these variables (see Figure 2). The probability distribution encoded by a dependency tree is given by

$$p(X_0, X_1, \ldots, X_{n-1}) = p(X_r) \prod_{i \neq r} p\left(X_i | X_{\pi(i)}\right),$$

where $(X_0, X_1, \ldots, X_{n-1})$ are random variables corresponding to the different string positions, r denotes the index of the variable at the root of the tree, $\pi(i)$ denotes the index of the parent of X_i, and the product runs over all indices $i \in \{0, 1, \ldots, n-1\}$ except for the root index r.

The parameters of the dependency tree are the probabilities $p(X_r = x_r)$ for all potential values x_r of the root X_r, and the conditional probabilities $p(X_i = x_i | X_{\pi(i)} = x_{\pi(i)})$ for all values x_i of each non-root variable X_i given any value $x_{\pi(i)}$ of X_i's parent, $X_{\pi(i)}$.

3.2 Learning Dependency Trees

The dependency tree model that most closely mimics the actual distribution of samples in the selected population with respect to the Kullback–Leibler divergence (Kullback & Leibler, 1951) is the one that maximizes the total mutual information between connected variables:

$$\arg\max_{r,\pi} \sum_{i \neq r} I(X_i; X_{\pi(i)}),$$

where $I(X_i; X_{\pi(i)})$ is the mutual information of X_i and $X_{\pi(i)}$:

$$I(X_i; X_{\pi(i)}) = \sum_x \sum_y p(X_i = x, X_{\pi(i)} = y) \log \frac{p(X_i = x, X_{\pi(i)} = y)}{p(X_i = x) p(X_{\pi(i)} = y)}.$$

Learning the structure of the best dependency tree model thus corresponds to finding a maximum spanning tree in a weighted graph where all pairs of nodes are connected and each edge is weighted by the mutual information between the variables it connects. To find the maximum spanning tree, a number of standard greedy algorithms can be used, such as the Prim's algorithm.

To learn parameters, the most likely values of all probabilities are computed from their frequencies m in the selected set of N solutions (Chow & Liu, 1968):

$$p(X_r = x) = \frac{m(X_r = x)}{N},$$

$$p(X_i = x | X_{\pi(i)} = y) = \frac{m(X_i = x, X_{\pi(i)} = y)}{m(X_{\pi(i)} = y)}$$

For more details on the model learning and sampling algorithms for dependency trees, please see Chow and Liu (1968) and Baluja and Davies (1997).

3.3 Standard Sampling

The most common approach to sampling decision trees and other similar models is *probabilistic logic sampling* (PLS) (Henrion, 1988). To sample candidate solutions according to the probability distribution encoded by the dependency tree, we first order variables so that each variable is preceded by its parent (the root is the first node). The variables are then sampled according to the computed ancestral ordering so that once we want to generate a variable that has a parent in the tree, its parent would have been generated already.

3.4 Sampling Permutations

Here we describe the modifications to the sampling algorithm for dependency trees that will allow us to generate permutations. The sampling algorithm is based on automatic adjustments to the parameters of the model to exclude samples that are not valid permutations (Bengoetxea, Larrañaga, Bloch, Perchant, & Boeres, 2000; Tsutsui, Pelikan, & Goldberg, 2006).

Analogically to sampling standard dependency trees, the algorithm starts by computing an ancestral ordering of the variables. Then, variables are sampled according to the computed ancestral ordering. However, in each iteration, the sampling is restricted to those values that have not yet been generated for this specific candidate solution. In this way, it is ensured that the resulting vector encodes a permutation.

To ensure that at each step of the sampling algorithm we use only allowed (not-yet-generated) values, we use flags to mark values that have already been generated for the current candidate solution. If the conditional probability of at least one of the remaining values is positive, the probabilities of these potential values are normalized by a constant factor to ensure that they sum to 1 and the value of the current variable is generated out of those possible values using the normalized conditional probabilities. If all possible values have 0 probability, we generate all possible values equally likely.

The above procedure can be improved by creating a random ancestral ordering for each new candidate solution; otherwise it can be expected that the sampling of the nodes located later in the ancestral ordering will be affected by a slight bias. Nonetheless, in our current implementation, we use a fixed ancestral ordering for the generation of the entire new population of points.

3.5 Niching via Restricted Tournaments

To ensure that useful diversity in the population is maintained, we use restricted tournament replacement (RTR) (Harik, 1995) to incorporate newly generated solutions into the original population. With RTR, to incorporate each new solution into the original population, a random subset of w (window size) solutions is first selected and the new solution is compared to every solution in the selected subset. The closest solution out of the subset competes with the new candidate solution; if the new candidate solution has better fitness, it replaces the most similar solution in the original population; otherwise, the new solution is discarded. The window size w is set as suggested in (Pelikan, Goldberg, & Lobo, 2002) as $w = \max\{n, N/20\}$ where n is the number of variables in the problem and N is the population size.

To make a distance metric used to identify the most similar candidate solutions in RTR more adequate for permutation problems, the distance metric is computed from the differences between the absolute positions of different permutation elements; more specifically, the distance of two permutations is defined as the sum of squares of the differences between the positions of each permutation element. Other distance measures may be used, for example, we may count the number of relative ordering constraints that are contradictory in the two individuals. Determining the most efficient replacement strategy remains a topic for future research.

4 Test Problems

We consider two basic types of permutation problems: (1) deceptive ordering problems and (2) the quadratic assignment problem.

4.1 Ordering Deceptive Problems

Decomposable problems with deceptive subproblems are often used to test the effectiveness of variation operators of evolutionary algorithms (Kargupta, Deb, & Goldberg, 1992; Knjazew & Goldberg, 2000; Bosman & Thierens, 2001b). Two types of deceptive problems have been proposed for the permutation domain: (1) relative ordering deceptive problem (Kargupta, Deb, & Goldberg, 1992; Bosman & Thierens, 2001b) and (2) absolute ordering deceptive problem (Kargupta, Deb, & Goldberg, 1992).

4.1.1 Relative Ordering Deceptive Problem

In the relative ordering deceptive problem (Kargupta, Deb, & Goldberg, 1992), the relative ordering of permutation elements is important. The fitness for a permutation is given by the sum of fitness values for subproblems, each containing 4 unique permutation elements. Only the relative order of the elements in each subproblem matters. Let us, without the loss of generality, denote the elements in the subproblem by 1, 2, 3, and 4; the fitness of the subproblem is given by

```
f(1 2 3 4) = 4.0   f(2 1 3 4) = 1.1   f(2 4 3 1) = 1.2   f(3 4 1 2) = 2.2
f(1 2 4 3) = 1.1   f(1 3 4 2) = 1.2   f(3 1 2 4) = 1.2   f(4 3 1 2) = 2.4
f(1 4 3 2) = 1.1   f(1 4 2 3) = 1.2   f(2 3 1 4) = 1.2   f(2 3 4 1) = 1.5
f(1 3 2 4) = 1.1   f(4 2 1 3) = 1.2   f(2 1 4 3) = 2.4   f(3 4 2 1) = 3.2
f(3 2 1 4) = 1.1   f(3 2 4 1) = 1.2   f(4 1 2 3) = 2.1   f(2 4 1 3) = 2.4
f(4 2 3 1) = 1.1   f(4 1 3 2) = 1.2   f(3 1 4 2) = 2.2   f(4 3 2 1) = 2.4
```

The problem can be divided into subproblems arbitrarily but for most operators applicable directly to permutations (unlike for random-key representations), it does not matter how we partition the problem.

4.1.2 Absolute Ordering Deceptive Problem

In the absolute ordering deceptive problem (Kargupta, Deb, & Goldberg, 1992), both relative and absolute ordering constraints matter. First of all, permutation elements in each subproblem should be located in the positions specified by the values in this subproblem. If some of the subproblem's elements are not within their partition, the fitness for the subproblem is equal to one half of the number of positions inside the partition. If all subproblem's positions are located in the correct partition, the fitness of a subproblem is computed similarly to the relative ordering problem:

```
f(1 2 3 4) = 4.0   f(2 1 3 4) = 1.8   f(2 4 3 1) = 2.0   f(3 4 1 2) = 2.6
f(1 2 4 3) = 1.8   f(1 3 4 2) = 2.0   f(3 1 2 4) = 2.0   f(4 3 1 2) = 2.6
f(1 4 3 2) = 1.8   f(1 4 2 3) = 2.0   f(2 3 1 4) = 2.0   f(2 3 4 1) = 2.6
f(1 3 2 4) = 1.8   f(4 2 1 3) = 2.0   f(2 1 4 3) = 2.6   f(3 4 2 1) = 3.3
f(3 2 1 4) = 1.8   f(3 2 4 1) = 2.0   f(4 1 2 3) = 2.6   f(2 4 1 3) = 2.6
f(4 2 3 1) = 1.8   f(4 1 3 2) = 2.0   f(3 1 4 2) = 2.6   f(4 3 2 1) = 2.6
```

The fitness is the sum of fitnesses of all subproblems, each occupying 4 permutation elements total. Similarly to the relative ordering problem, for most operators manipulating directly permutations it does not matter how we partition the problem.

In both the relative ordering problem and the absolute ordering problem, the subproblems are deceptive and they cannot be further decomposed. If variation operators do not process the subproblems effectively, the search will be deceived away from the global optimum. In the relative ordering problem, only the relative ordering constraints between elements in each subproblem matter, while in the absolute ordering problem the positions in each subproblems must be located in specific problem partitions as well.

4.2 Quadratic Assignment Problem

In the quadratic assignment problem (QAP) (Koopmans & Beckmann, 1957), we are given n locations and n facilities and the task is to assign the facilities to the locations to minimize the

cost. For each pair of locations i and j, we are given their distance $d(i,j)$. For each pair of facilities i and j, we are given the flow $f(i,j)$ between these facilities (regardless of their locations). The cost of each assignment of facilities to locations is given as the sum over all pairs of locations of the distance of these locations and the flow between the facilities assigned to these locations.

Assignments of facilities to locations can be encoded as permutations; the facility at location i is given by the value at position i of the permutation. The cost for an assignment $(X_0, X_1, \ldots, X_{n-1})$ that is to be minimized is given by

$$cost(X_0, X_1, \ldots, X_{n-1}) = \sum_{i \neq j} d(i,j) f(X_i, X_j).$$

To improve efficiency of evolutionary algorithms on QAP, we use a local search algorithm based on 2-opt. In each iteration of the local search, a pair of facilities that maximizes the decrease in the cost is exchanged and the step is repeated until it becomes impossible to improve the cost by exchanging a pair of facilities.

QAP is NP-complete and many important problems can be formulated as instances of QAP, from the design of storage-and-retrieval devices (Polak, 2005) to microarray placement problems (de Carvalho Jr. & Rahmann, 2006).

The QAP instances used in this paper were obtained from QAPLIB (Burkard, Karisch, & Rendl, 1997), Érick Taillard's web page on QAP (Taillard, 2006), and the web page of Sergio A. de Carvalho Jr. with QAP instances for the microarray placement problem (de Carvalho Jr. & Rahmann, 2006). For most instances, globally optimal solutions are known and the goal of the experiments was to only measure performance of dtEDA and other compared methods on these instances. However, in the QAP instances for microarray data, the published solutions were only locally optimal and for many instances we have obtained solutions of lower cost than the published results (de Carvalho Jr. & Rahmann, 2006).

5 Experiments

5.1 Description of Experiments

In all experiments, binary tournament selection is used to obtain the selected population of points and the new candidate solutions are incorporated into the original population using RTR with window size $w = \max\{n, N/20\}$.

In addition to dtEDA, the we include the results for the genetic algorithm with two common two-parent recombination operators for permutations: (1) the partially matched crossover (PMX) (Goldberg & Lingle, 1985; Goldberg, 1989), and (2) the linear order crossover (LOX) (Falkenauer & Bouffouix, 1991). The probability of crossover is set to 0.6 in both cases. The genetic algorithm with PMX and LOX uses the same selection and replacement operators as dtEDA.

For all problems with a known global optimum and all algorithms, the population size was determined automatically using the bisection method (Sastry, 2001) so that a minimum population size to ensure a reliable convergence in 10 out of 10 independent runs is used.

For QAP we also included the robust tabu search from Taillard (2006), which represents a popular approach to solving QAP based on local search. The tabu search is used with the default settings of the implementation in Taillard (2006).

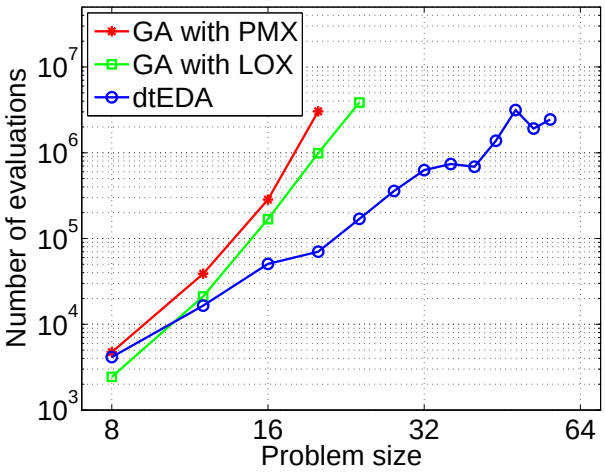

Figure 3: Results on absolute ordering problem.

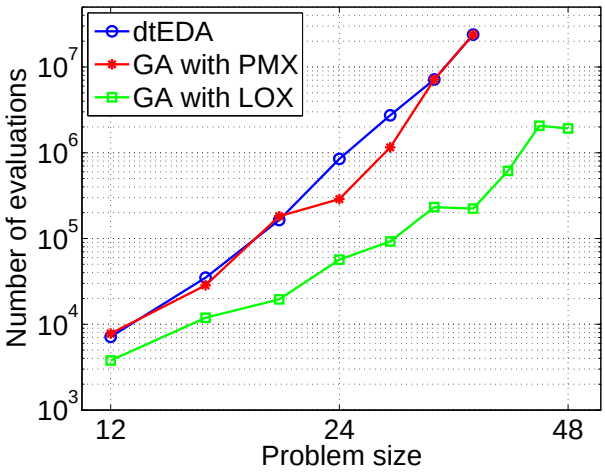

Figure 4: Results on relative ordering problem.

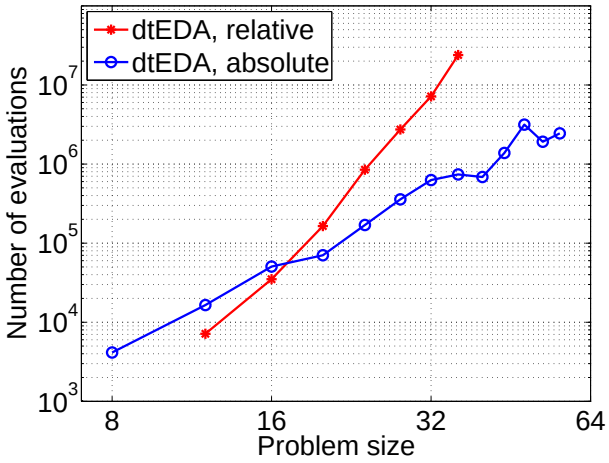

Figure 5: dtEDA performance on deceptive absolute and relative ordering problems.

5.2 Results on Deceptive Ordering Problems

The results on the relative ordering problem and the absolute ordering problem are shown in Figures 3, 4, and 5. The results indicate that dtEDA significantly outperforms GA with both PMX and LOX on the absolute ordering problem (see Figure 3). On the other hand, on the relative ordering problem, dtEDA performs similarly to GA with PMX but both these methods are outperformed by GA with LOX (see Figure 4). The comparison of dtEDA performance on the two deceptive problems confirms that dtEDA performs better for the absolute ordering problem (see Figure 5).

The results on the deceptive problems indicate that dtEDA performs better in problems where absolute ordering of permutation is important than in problems where only relative ordering is important. We believe that the reason for this behavior is that dtEDA represents each position in a permutation with a separate variable, which leads to a relatively inefficient representation of regularities in problems where only relative ordering matters.

5.3 Results on QAP

We performed a large number of experiments on many publicly available instances of QAP to analyze the behavior of the compared methods and their scalability. This section summarizes and discusses the obtained results. Since all algorithms incorporate the local search based on the exchanges of pairs of permutation elements in some way, computational complexity of all methods is measured by the number of steps of the local searcher until the global optimum has been reached. To obtain a more accurate estimate of the time complexity of the robust tabu search, we run the algorithm 10 times and average the number of steps until the global optimum has been reached.

Figure 6 shows the results on symmetric uniformly generated random QAP instances `tai10a` to `tai35a` from Taillard (1991). Figure 6 shows the results on asymmetric uniformly generated QAP instances `tai10b` to `tai25b` from (Taillard, 1991). Figure 8 shows the results on the structured symmetric QAP instances `tai27e01-10` to `tai75e01-10` (10 instances for each problem size) from Drezner, Hahn, and Taillard (2002). Figure 9 shows the results on the random instances `lipa10` to `lipa60` from Li and Pardalos (1992). Figure 10 shows the results on the random instances `lipb10` to `lipb80` from Li and Pardalos (1992).

Most results indicate the robust tabu search performs better than dtEDA, GA with PMX, and GA with LOX. The structured, symmetric instances `tai27e` to `tai75e` are the only case where the algorithms based primarily on recombination outperform the robust tabu search. These results are in agreement with the hypothesis that most publicly available random QAP instances do not contain much structure and represent a challenge for metaheuristics that exploit some form of the problem structure (Angel & Zissimopoulos, 2001).

GA with PMX outperforms GA with LOX in most large QAP instances; the primary reason for this result is that LOX focuses primarily on preserving relative ordering constraints while PMX is also capable of encoding absolute positions, which are important in QAP. For large problems, dtEDA outperforms GA with PMX and GA with LOX in most cases, although the differences do not look very significant in the log-log plots.

In addition to the standard random QAP instances discussed above, we applied all compared methods to the QAP instances created by transforming the microarray placement problem to QAP (de Carvalho Jr. & Rahmann, 2006). To verify previously published results, we ran dtEDA and GA on these instances with a large population size $N = 10000$, confirming that these problem instances are the most difficult ones out of all tested QAP instances. For the conflict index minimization problems, dtEDA, GA with PMX, and GA with LOX were able to obtain solutions with lower cost than those published at `http://gi.cebitec.uni-bielefeld.de/assb/chiplayout/qap`. Some of these results are summarized below:

Size	GA-TS	GRASP	dtEDA
6×6	169,016,907	169,925,219	168,705,477
7×7	237,077,377	238,859,844	236,355,034
8×8	326,696,412	327,770,071	326,251,579
9×9	428,682,120	434,317,170	427,852,554

In the above table, the best published results are shown in the columns GA-TS (GA with tabu search from Rodriguez, MacPhee, Bonham, Horton, and Bhavsar (2004)) and GRASP-PR (GRASP with path relinking from Oliveira, Pardalos, and Resende (2004)); the last column lists the results obtained with dtEDA combined with local search based on the 2-opt heuristic. The two variants of

the standard GA discussed in this paper were able to duplicate some of the results obtained with dtEDA, but in no cases were they able to improve upon them.

6 Future Work

One of the topics for future work is to analyze dtEDA and similar EDAs for permutation problems on other important classes of such problems, such as p-median (Cristofides, 1975; Swamy & Thulasiraman, 1991; Taillard, 1998). Other types of probabilistic models can be implemented to work directly with permutations (Bengoetxea, Larrañaga, Bloch, Perchant, & Boeres, 2000), such as Bayesian networks and marginal product models; an important topic for future work is to investigate when more complex models pay off and when they are unnecessary. Additionally, various modifications can be incorporated into EDAs for permutations in a straightforward manner, such as the templates (Tsutsui, Goldberg, & Pelikan, 2002), to improve performance of dtEDA and other compared methods. Yet another important direction for future research is to obtain a better understanding of problem difficulty of the quadratic assignment problem and other classes of problems in the permutation domain.

7 Summary and Conclusions

This paper described and analyzed an estimation of distribution algorithm based on dependency tree models (dtEDA), which can explicitly encode probabilistic models for permutations. dtEDA was tested on deceptive ordering problems and a number of instances of the quadratic assignment problem. The results on deceptive ordering problems indicated that dtEDA performs better on problems where both the absolute and relative ordering constraints are important. In the quadratic assignment problem, dtEDA outperformed standard recombination operators of genetic algorithms, although in many random classes of the quadratic assignment problem, the robust tabu search performed best. Best results were obtained in the most difficult instances of the quadratic assignment problem created by transforming the problem of aligning microarray data, where dtEDA and other approaches based on evolutionary algorithms provided better solutions than those published in the past. While the differences between dtEDA and other evolutionary algorithms for permutations are not as significant as in the classes of nearly decomposable and hierarchical problems, using EDAs for permutation problems appears to pay off in most difficult problems and proves to be a robust approach to optimization in the permutation domains.

Acknowledgments

This project was sponsored by the National Science Foundation under CAREER grant ECS-0547013, by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, and by the University of Missouri in St. Louis through the High Performance Computing Collaboratory sponsored by Information Technology Services, and the Research Award and Research Board programs.

The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. Most experiments were completed at the Beowulf cluster at the University of Missouri–St. Louis.

References

Angel, E., & Zissimopoulos, V. (2001). On the landscape ruggedness of the quadratic assignment problem. *Theoretical computer science*, *263*(1-2), 159–172.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Baluja, S., & Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. *Proceedings of the International Conference on Machine Learning*, 30–38.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, *6*(2), 154–160.

Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., & Boeres, C. (2000). Inexact graph matching using learning and simulation of Bayesian networks.

Bosman, P. A. N., & Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithms within the IDEA framework. *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 197–200.

Bosman, P. A. N., & Thierens, D. (2001a). Crossing the road to efficient ideas for permutation problems. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 219–226.

Bosman, P. A. N., & Thierens, D. (2001b). New IDEAs and more ICE by learning and using unconditional permutation factorizations. In *Late-Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO 2001)* (pp. 13–23).

Bryan, N., Alice, A., & Smith, E. (1997). Random keys genetic algorithm with adaptive penalty function for optimization of constrained facility layout problems. *Proceedings of the International Conference on Evolutionary Computation (ICEC-97)*, 407–411.

Burkard, R., Karisch, S., & Rendl, F. (1997). QAPLIB – A quadratic assignment problem library. *Journal of Global Optimization*, *10*, 391–403.

Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, *14*, 462–467.

Cristofides, N. (1975). *Graph theory. An algorithm approach.* New York: Academic Press.

de Carvalho Jr., S. A., & Rahmann, S. (2006). Microarray layout as a quadratic assignment problem. *German Conference on Bioinformatics (GCB).*

Drezner, Z., Hahn, P., & Taillard, E. D. (2002). *A study of qap instances that are difficult for meta-heuristic methods* (Technical Report). INA, Yverdon-les-Bains.

Falkenauer, E., & Bouffouix, S. (1991). A genetic algorithm for job shop. *Proceedings 1991 IEEE International Conference on Robotics and Automation*, 824–829.

Gambardella, L.-M., Taillard, É. D., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problems. *Journal of the Operational Research Society*, *50*, 167–176.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E., & Lingle, Jr., R. (1985). Alleles, loci, and the traveling salesman problem. *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, 154–159.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *Proceedings of the International Conference on Genetic Algorithms (ICGA-95)*, 24–31.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence* (pp. 149–163). Amsterdam, London, New York: Elsevier.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Kargupta, H., Deb, K., & Goldberg, D. E. (1992). Ordering genetic algorithms and deception. *Parallel Problem Solving from Nature II*, 49–58.

Knjazew, D., & Goldberg, D. E. (2000). *OMEGA - Ordering messy GA : Solving permutation problems with the fast messy genetic algorithm and random keys* (IlliGAL Report No. 2000004). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Koopmans, T. C., & Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica*, *25*, 53–76.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *Annals of Math. Stats.*, *22*, 79–86.

Larrañaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston, MA: Kluwer.

Li, Y., & Pardalos, P. M. (1992). Generating quadratic assignment test problems with known optimal permutations. *Computational Optimization and Applications*, *1*, 163–184.

Merkle, D., & Middendorf, M. (2005). On solving permutation scheduling problems with ant colony optimization. *International Journal of Systems Science*, *36*(5), 255–266.

Mühlenbein, H., & Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, *7*(4), 353–376.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, 178–187.

Oliveira, Pardalos, & Resende (2004). GRASP with path-relinking for the quadratic assignment problem. In *International Workshop on Experimental and Efficient Algorithms (WEA), LNCS*,

Pelikan, M. (2005). *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Springer-Verlag.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20. Also IlliGAL Report No. 99018.

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. *Advances in Soft Computing—Engineering Design and Manufacturing*, 521–535.

Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.) (2006). *Scalable optimization via probabilistic modeling: From algorithms to applications*. Springer-Verlag.

Pelikan, M., Sastry, K., & Goldberg, D. E. (2002). Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning*, *31*(3), 221–258. Also IlliGAL Report No. 2001029.

Polak, G. G. (2005). On a special case of the quadratic assignment problem with an application to storage-and-retrieval devices. *Annals of Operations Research*, *138*(1), 223–233.

Robles, V., de Miguel, P., & Larrañaga, P. (2001). Solving the traveling salesman problem with edas. Kluwer.

Rodriguez, J. M., MacPhee, F. C., Bonham, D. J., Horton, J. D., & Bhavsar, V. C. (2004). Best permutations for the dynamic plant layout problem. *Proceedings of the 12th International Conference on Advances in Computing and Communications (ADCOM 2004)*, 173–178.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms* (2nd ed.). Springer.

Rothlauf, F., Goldberg, D. E., & Heinzl, A. (2002). Network random keys—tree representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation*, *10*(1), 75–97.

Sastry, K. (2001). *Evaluation-relaxation schemes for genetic and evolutionary algorithms*. Master's thesis, University of Illinois at Urbana-Champaign, Department of General Engineering, Urbana, IL. Also IlliGAL Report No. 2002004.

Stutzle, T. (1998). An ant approach to the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT 98)* (pp. 1560–1564).

Swamy, M., & Thulasiraman, K. (1991). *Graphs, networks and algorithms*. New York: Wiley.

Taillard, E. (1991). Robust tabu search for the quadratic assignment problem. *Parallel Computing*, *17*, 443–455.

Taillard, E. D. (1998). *FANT: Fast ant system* (Technical Report IDSIA-46-98). IDSIA.

Taillard, É. D. (2006). Éric d. Taillard's home page. http://ina2.eivd.ch/collaborateurs/etd/.

Tsutsui, S., Goldberg, D. E., & Pelikan, M. (2002). *Solving sequence problems by building and sampling edge histograms* (IlliGAL Report No. 2002024). Urbana, IL: Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Tsutsui, S., Pelikan, M., & Goldberg, D. E. (2006). *Node histogram vs. edge histogram: A comparison of pmbgas in permutation domains* (MEDAL Report No. 2006009). Missouri Estimation of Distribution Algorithms Laboratory, University of Missour–St. Louis, St. Louis, MO.

Zhang, Q., Sun, J., Tsang, E. P. K., & Ford, J. (2003). Combination of guided local search and estimation of distribution algorithm for solving quadratic assignment problem. In *Workshop Proceedings at the Genetic and Evolutionary Computation Conference (GECCO-2003)* (pp. 42–48).

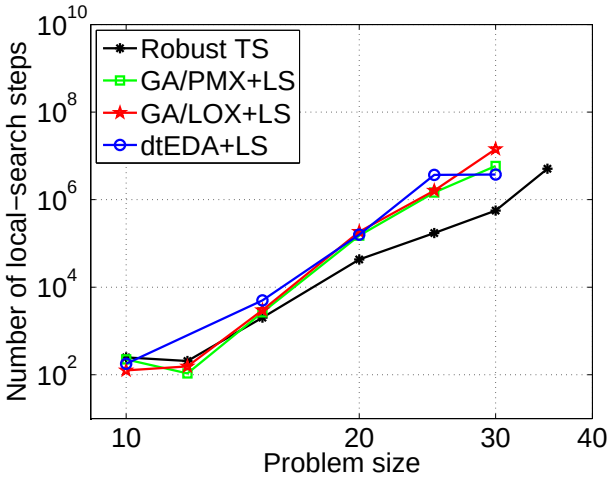

Figure 6: QAP instances `tai10a` to `tai35a`.

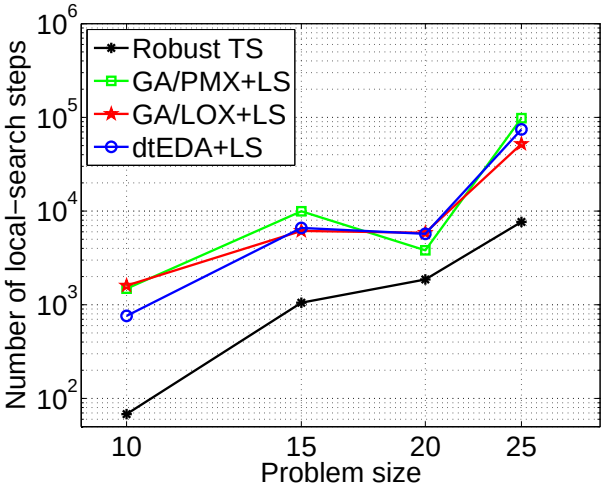

Figure 7: QAP instances `tai10b` to `tai25b`.

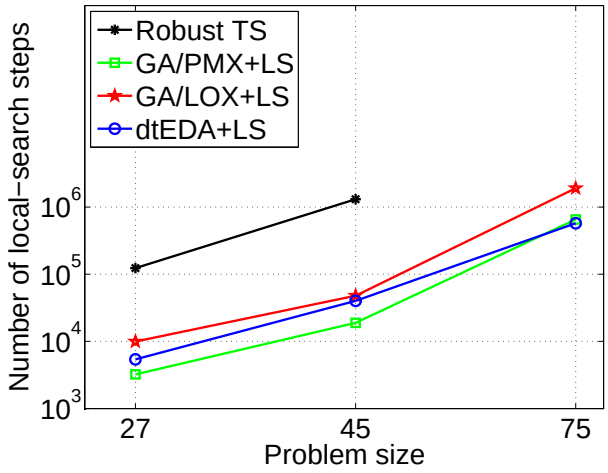

Figure 8: QAP instances `tai27e` to `tai75e`.

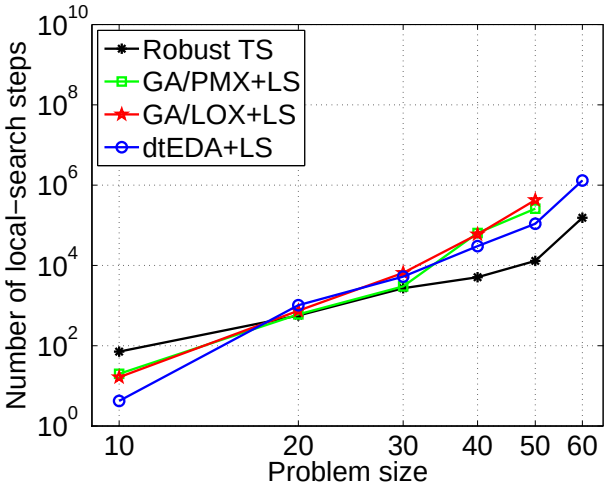

Figure 9: QAP instances `lipa10` to `lipa60`.

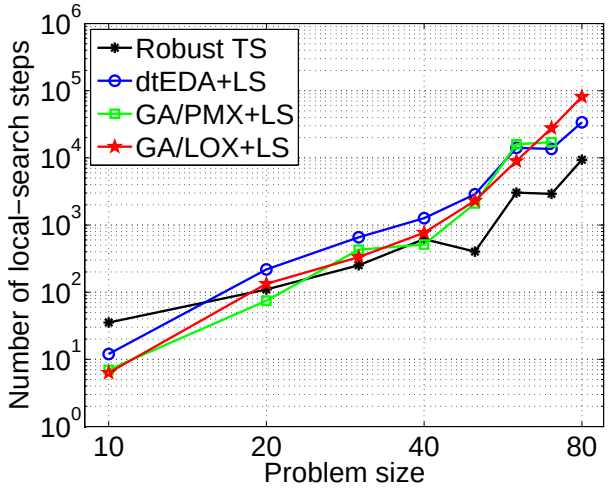

Figure 10: QAP instances `lipb10` to `lipb80`.