

Greedy Like Algorithms for the Traveling Salesman and Multidimensional Assignment Problems

Gregory Gutin Daniel Karapetyan

1 Introduction

Majority of chapters of this book show usefulness of greedy like algorithms for solving various combinatorial optimization problems. The aim of this chapter is to warn the reader that not always a greedy like approach is a good option and, in certain cases, it is a very bad option being sometimes among the worst possible options. Our message is not a discouragement from using greedy like algorithms altogether; we think that for every combinatorial optimization problem of importance, researchers and practitioners should simply investigate the appropriateness of greedy like algorithms and the existence of better alternatives to them (considering both quality of solution and running time). In many cases, especially when the running time must be very short, the conclusion may still be that the most practical of known approaches is a greedy like algorithm.

The Traveling Salesman and Multidimensional Assignment Problems are optimization problems for which greedy like approaches are usually not very successful. We demonstrate this by providing both theoretical and experimental results on greedy like algorithms as well as on some other algorithms that produce (in theory and/or in experiments) much better results without spending significantly more time.

There are some general theoretical results that indicate that there are, in fact, many combinatorial optimization problems for which greedy like algorithms are not the best option even among fast construction heuristics, see, e.g., [3, 5, 17]. We will not consider these general results in order to avoid most mathematical details

that are not necessary for understanding the results of this chapter. For this reason we will not give proofs here apart from two simple proofs: that of Theorem 8 which shows that some instances on which the greedy algorithm fails are not exotic in a sense and that of Theorem 11 since Theorem 11 is a new result.

It is not a trivial question whether a certain algorithm is greedy like or not. In the next section we define an independence system and give the classic definition of the greedy algorithm for such a system. We extend this definition to so-called greedy type algorithms that include such well-known algorithms as the Prim's algorithm for the minimum spanning tree problem and the nearest neighbor algorithm for the traveling salesman problem. We use the term 'greedy like' in an informal way and we include in this class simple and fast construction heuristics that seem to us to be of greedy nature. Unfortunately, no formal definition exists for the wide family of greedy like algorithms and one can understand the difficulty to formally classify such algorithms by, for example, considering local search algorithms which find the best solution in each neighborhood they search. Intuitively, it is clear that such local search algorithms are not greedy yet their every search is greedy in a sense.

In the next section, we give most of terminology and notation used in this chapter. Several results on theoretical performance of greedy like algorithms for the Traveling Salesman and Multidimensional Assignment Problems are discussed in Sections 3 and 4, respectively. Experimental results on greedy like algorithms for the Traveling Salesman and Multidimensional Assignment Problems are given and analyzed in Sections 5 and 6, respectively.

2 Terminology and Notation

The *Asymmetric Traveling Salesman Problem (ATSP)* is the problem of computing a minimum weight tour (Hamilton directed cycle) passing through every vertex in a weighted complete digraph K_n^* on n vertices. The *Symmetric TSP (STSP)* is the same problem, but on a weighted complete undirected graph K_n . When a certain fact holds for both ATSP and STSP, we will simply speak of *TSP*. We often assume that the vertices of K_n^* and K_n are $1, 2, \dots, n$ and often refer to the weight $w(ij)$ of an edge of K_n^* (or K_n) as the *distance* from i to j . TSP has a large number of applications, see, e.g., the two recent books [1, 13] on TSP.

The Multidimensional Assignment Problem (MAP) (abbreviated *s*-AP in the

case of s dimensions) is a well-known optimization problem with a host of applications (see, e.g., [2, 6, 7] for ‘classic’ applications and [4, 25] for recent applications in solving systems of polynomial equations and centralized multisensor multitarget tracking). In fact, several applications described in [4, 6, 25] naturally require the use of s -AP for values of s larger than 3.

For a fixed $s \geq 2$, the s -AP is stated as follows. Let $X_1 = X_2 = \dots = X_s = \{1, 2, \dots, n\}$. We will consider only vectors that belong to the Cartesian product $X = X_1 \times X_2 \times \dots \times X_s$. Each vector $e \in X$ is assigned a non-negative integral weight $w(e)$. For a vector $e \in X$, the component e_j denotes its j th coordinate, i.e., $e_j \in X_j$. A collection A of $t \leq n$ vectors e^1, e^2, \dots, e^t is a (*feasible*) *partial assignment* if $e_j^i \neq e_j^k$ holds for each $i \neq k$ and $j \in \{1, 2, \dots, s\}$. The *weight* of a partial assignment A is $w(A) = \sum_{i=1}^t w(e^i)$. An *assignment* (or *full assignment*) is a partial assignment with n vectors. The objective of s -AP is to find an assignment of minimum weight.

Let \mathcal{P} be a combinatorial optimization problem and let \mathcal{H} be a heuristic for \mathcal{P} . The *domination number* $\text{domn}(\mathcal{H}, \mathcal{I})$ of \mathcal{H} for an instance \mathcal{I} of \mathcal{P} is the number of solutions of \mathcal{I} that are not better than the solution s produced by \mathcal{H} including s itself. For example, consider an instance \mathcal{T} of the STSP on 5 vertices. Suppose that the weights of tours in \mathcal{T} are 2, 5, 5, 6, 6, 9, 9, 11, 11, 12, 12, 15 (every instance of STSP on 5 vertices has 12 tours) and suppose that the greedy algorithm computes the tour T of weight 6. Then $\text{domn}(\text{greedy}, \mathcal{T}) = 9$. In general, if $\text{domn}(\mathcal{H}, \mathcal{I})$ equals the number of solutions in \mathcal{I} , then \mathcal{H} finds an optimal solution for \mathcal{I} . If $\text{domn}(\mathcal{H}, \mathcal{I}) = 1$, then the solution found by \mathcal{H} for \mathcal{I} is the unique worst possible one. The *domination number* $\text{domn}(\mathcal{H}, n)$ of \mathcal{H} is the minimum of $\text{domn}(\mathcal{H}, \mathcal{I})$ over all instances \mathcal{I} of size n .

An *independence system* is a pair consisting of a finite set E and a family \mathcal{F} of subsets (called *independent sets*) of E such that (I1) and (I2) are satisfied.

(I1) the empty set is in \mathcal{F} ;

(I2) If $X \in \mathcal{F}$ and Y is a subset of X , then $Y \in \mathcal{F}$.

All maximal sets of \mathcal{F} are called *bases* (or, *feasible solutions*).

Many combinatorial optimization problems can be formulated as follows. We are given an independence system (E, \mathcal{F}) , a set $W \subseteq \mathbb{Z}_+$ and a weight function w that assigns a weight $w(e) \in W$ to every element of E (\mathbb{Z}_+ is the set of non-negative integers). The weight $w(S)$ of $S \in \mathcal{F}$ is defined as the sum of the weights of the

elements of S . It is required to find a base $B \in \mathcal{F}$ of minimum weight. We will consider only such problems and call them the (E, \mathcal{F}, W) -*optimization problems*.

ATSP is an $(E, \mathcal{F}, \mathbb{Z}_+)$ -optimization problem, where E is the set of arcs of the complete digraph K_n^* and $\mathcal{F} = \{B \subseteq H : H \in \mathcal{H}\}$, where \mathcal{H} is the set of Hamilton directed cycles of K_n^* . MAP is also an $(E, \mathcal{F}, \mathbb{Z}_+)$ -optimization problem, where E is the set of all vectors and \mathcal{F} is the set of all partial assignments.

If $S \in \mathcal{F}$, then let $I(S) = \{x : S \cup \{x\} \in \mathcal{F}\} \setminus S$. This means that $I(S)$ consists of those elements from $E \setminus S$, which can be added to S , in order to have an independent set of size $|S| + 1$. Note that by (I2) $I(S) \neq \emptyset$ for every independent set S which is not a base.

The *Greedy Algorithm* (Greedy) tries to construct a minimum weight base as follows: it starts from an empty set X , and at every step it takes the current set X and adds to it a minimum weight element $e \in I(X)$, the algorithm stops when a base is built. We assume that the greedy algorithm may choose any element among equally weighted elements in $I(X)$. Thus, when we say that the greedy algorithm *may construct* a base B , we mean that B is built provided the appropriate choices between elements of the same weight are made.

Greedy type algorithms were introduced in [14]. They include the nearest neighbor algorithm for TSP and are defined as follows. A *greedy type* algorithm \mathcal{H} is similar to the greedy algorithm: start with the partial solution $X = \emptyset$; and then repeatedly add to X an element of minimum weight in $I_{\mathcal{H}}(X)$ (ties are broken arbitrarily) until X is a base of \mathcal{F} , where $I_{\mathcal{H}}(X)$ is a subset of $I(X)$ that does not depend on the cost function c , but only on the independence system (E, \mathcal{F}) and the set X . Moreover, $I_{\mathcal{H}}(X)$ is non-empty if $I(X) \neq \emptyset$, a condition that guarantees that \mathcal{H} always outputs a base.

3 Theoretical Performance of Greedy Like Algorithms for TSP

The main practical message of this and the next section is that one should be careful while using the classical greedy algorithm and its variations in combinatorial optimization: there are many instances of combinatorial optimization problems for which such algorithms will produce the unique worst possible solution. Moreover,

this is true for several well-known optimization problems and the corresponding instances are not exotic, in a sense. This means that not always the paradigm of greedy optimization provides any meaningful optimization at all.

The first results of the kind mentioned in the previous paragraph were obtained in [19]:

Theorem 1. *For each $n \geq 2$ there is an instance of ATSP for which the Greedy Algorithm finds the unique worst possible tour.*

Gutin, Yeo and Zverovitch [19] proved Theorem 1 also for the *Nearest Neighbor* (NN) algorithm: start from an arbitrary vertex i_1 and go to a vertex $i_2 \neq i_1$ with shortest distance from i_1 ; when in a vertex i_k , $k < n$, go to a vertex i_{k+1} with shortest distance from i_k among vertices not in the set $\{i_1, i_2, \dots, i_{k-1}\}$. The proof for NN is correct for both ATSP and STSP. The proof of Theorem 1 itself (for Greedy) cannot be extended to STSP, but Theorem 1 holds also for STSP [18].

The Greedy and NN algorithms are special cases of greedy type algorithms and Bendall and Margot [5] proved the following result that generalizes all the results above.

Theorem 2. *Let \mathcal{A} be a TSP greedy type algorithm. For each $n \geq 3$ there is an instance of TSP for which \mathcal{A} finds the unique worst possible tour.*

At this stage the reader may ask the following natural question: ‘Perhaps, it is true that every TSP heuristic has the domination number equal 1?’ The answer is negative. In fact, there are many TSP heuristics (see, e.g., [15, 23]) which, for every instance T of TSP with n vertices ($n \geq 3$), produce a tour that is no longer than the average length of a tour in T . Among these heuristics there are several fast construction heuristics. Thus, we can apply the following theorem to many TSP heuristics (we formulate this theorem for ATSP, but an almost identical result by Rublineckii [26] is known for STSP, see [13]):

Theorem 3. *Let \mathcal{H} be an ATSP heuristics that, for every instance T of ATSP on $n \geq 2$ vertices, produces a tour that is no longer than the average length of a tour in T . Then the domination number of \mathcal{H} is at least $(n - 2)!$ for each $n \neq 6$.*

This theorem was first proved by Sarvanov [27] for odd values of n and by Gutin and Yeo [15] for even values of n .

Sometimes, we are interested in TSP with only restricted range of weights. The following two results for this variation of TSP were obtained by Bang-Jensen, Gutin and Yeo [3].

Theorem 4. *Consider STSP as an (E, \mathcal{H}, W) -optimization problem.*

- (a) *If $n \geq 4$ and $|W| \leq \lfloor \frac{n-1}{2} \rfloor$, then the greedy algorithm never produces the unique worst possible base (i.e., tour).*
- (b) *If $n \geq 3$, $r \geq n-1$ and $W = \{1, 2, \dots, r\}$, then there exists a weight function $w : E \rightarrow \{1, 2, \dots, r\}$ such that the greedy algorithm may produce the unique worst possible base (i.e., tour).*

Theorem 5. *Consider ATSP as an (E, \mathcal{H}, W) -optimization problems. Let $n \geq 3$.*

- (a) *If $|W| \leq \lfloor \frac{n-1}{2} \rfloor$, then the greedy algorithm never produces the unique worst possible base (i.e., tour).*
- (b) *For every $r \geq \lceil \frac{n+1}{2} \rceil$ there exists a weight function $w : E(K_n^*) \rightarrow \{1, 2, \dots, r\}$ such that the greedy algorithm may produce the unique worst possible base (i.e., tour).*

Notice that the above-mentioned theorems can be proved as corollaries of general results that hold for many (E, \mathcal{H}, W) -optimization problems, see, e.g., [3, 5, 16].

Another ATSP greedy like heuristic, max-regret-fc (fc abbreviates First Coordinate), was first introduced by Ghosh et al. [8]. Extensive computational experiments in [8] demonstrated a clear superiority of max-regret-fc over the greedy algorithm and several other construction heuristics from [9]. Therefore, the result of Theorem 6 obtained by Gutin, Goldengorin and Huang [11] was somewhat unexpected.

Let Q be a collection of disjoint directed paths in K_n^* and let $V = V(K_n^*) = \{1, 2, \dots, n\}$. An arc $a = ij$ is a *feasible addition* to Q if $Q \cup \{a\}$ is either a collection of disjoint paths or a tour in K_n^* . Consider the following two ATSP heuristics: max-regret-fc and max-regret.

The heuristic *max-regret-fc* proceeds as follows. Set $W = T = \emptyset$. While $W \neq V$ do the following: For each $i \in V \setminus W$, compute two lightest arcs ij and ik that are feasible additions to T , and compute the difference $\Delta_i = |w(ij) - w(ik)|$.

For $i \in V \setminus W$ with maximum Δ_i choose the lightest arc ij , which is a feasible addition to T and add ij to M and i to W .

The heuristic *max-regret* proceeds as follows. Set $W^+ = W^- = T = \emptyset$. While $W^+ \neq V$ do the following: For each $i \in V \setminus W^+$, compute two lightest arcs ij and ik that are feasible additions to T , and compute the difference $\Delta_i^+ = |w(ij) - w(ik)|$; for each $i \in V \setminus W^-$, compute two lightest arcs ji and ki that are feasible additions to T , and compute the difference $\Delta_i^- = |w(ji) - w(ki)|$. Compute $i' \in V \setminus W^+$ with maximum $\Delta_{i'}^+$ and $i'' \in V \setminus W^-$ with maximum $\Delta_{i''}^-$. If $\Delta_{i'}^+ \geq \Delta_{i''}^-$ choose the lightest arc $i'j'$, which is a feasible addition to T and add $i'j'$ to M , i' to W^+ and j' to W^- . Otherwise, choose the lightest arc $j''i''$, which is a feasible addition to T and add $j''i''$ to M , i'' to W^- and j'' to W^+ .

Notice that in max-regret-fc, if $|V \setminus W| = 1$ we set $\Delta_i = 0$. A similar remark applies to max-regret.

Theorem 6. *The domination number of both max-regret-fc and max-regret equals 1 for each $n \geq 2$.*

4 Theoretical Performance of Greedy Like Algorithms for MAP

In this section, we will first prove that the greedy algorithm for s-AP is of domination number 1. The proof shows that the greedy algorithm fails on instances that cannot be called ‘exotic’ in the sense that they do not have very large weights. For our proof we need the following definitions and lemma.

A vector h is *backward* if $\min\{h_i : 2 \leq i \leq s\} < h_1$; a vector h is *horizontal* if $h_1 = h_2 = \dots = h_s$. A vector is *forward* if it is not horizontal or backward.

Lemma 7. *Let F be an assignment of s-AP ($s \geq 2$). Either all vectors of F are horizontal or F contains a backward vector.*

Proof: Let $F = \{f^1, f^2, \dots, f^n\}$, where $f_1^i = i$ for each $1 \leq i \leq n$. Assume that not every vector of F is horizontal. We show that F has a backward vector. Suppose it is not true. Then F has a forward vector f^i . Thus, there is a subscript j such that $f_j^i > i$. By the pigeonhole principle, there exists a superscript $k > i$ such that $f_j^k \leq i$, i.e., f^k is backward; a contradiction. \square

Theorem 8. For each $s \geq 2$, $n \geq 2$, there exists an instance of s -AP for which Greedy will find the unique worst possible assignment.

Proof: Consider some $M > n$ and let $E = \{e^1, e^2, \dots, e^n\}$, where $e^i = (i, i, \dots, i)$ for every $1 \leq i \leq n$. We define the required instance \mathcal{I} as follows: $w(e^i) = iM$ for each $1 \leq i \leq n$ and, for each $f \notin E$, $w(f) = \min\{f_i : 1 \leq i \leq s\} \cdot M + 1$.

Observe that Greedy will construct E . Let $F = \{f^1, f^2, \dots, f^n\}$ be any other assignment, where $f_1^i = i$ for each $1 \leq i \leq n$. By Lemma 7, F has a backward vector f^k . Notice that

$$w(f^k) \leq (k-1)M + 1 \quad (1)$$

By the definition of the weights and (1),

$$\begin{aligned} w(F) &= \sum_{i=1}^n w(f^i) = \sum_{i \neq k} w(f^i) + w(f^k) \\ &\leq \sum_{i \neq k} (iM + 1) + (k-1)M + 1 \\ &= \sum_{i=1}^n iM + n - M \\ &< \sum_{i=1}^n iM = w(E). \end{aligned}$$

□

One can consider various greedy type algorithms for s -AP. One natural algorithm of this kind proceeds as follows: At the i th iteration, $i = 1, 2, \dots, n$ choose a vector e^i of minimum weight such that $e_1^i = i$ and $\{e^1, e^2, \dots, e^i\}$ is a partial assignment. We call this algorithm the *First Coordinate Fixing* (FCF) heuristic. A simple modification of the proof of the first half of Theorem 10 in [5] shows the following:

Theorem 9. For every $n \geq 1$, $s \geq 2$, there is an instance of s -AP and for every greedy type algorithm \mathcal{H} for s -AP, there is an instance I of s -AP for which \mathcal{H} finds the unique worst possible assignment.

In Section 3, we considered the max-regret-fc and max-regret heuristics. In fact, max-regret was first introduced for 3-AP by Balas and Saltzman [2]. The s -AP heuristic *max-regret* proceeds as follows. We start from the empty partial assignment A and, at every iteration, we consider a partial assignment A . Set $V_d = \{1, 2, \dots, n\}$ for each $1 \leq d \leq s$. For each dimension d and each value $v \in V_d$ consider every vector $e \in X'$ such that $e_d = v$, where $X' \subset X$ is a set of feasible additional vectors, i.e., $A \cup \{e\}$ is a feasible partial assignment if $e \in X'$. If $X' \neq \emptyset$, find two vectors e^1_{\min} and e^2_{\min} in the considered subset $Y_{d,v} = \{e \in X' : e_d = v\}$ such that $e^1_{\min} = \operatorname{argmin}_{e \in Y_{d,v}} w(e)$, and $e^2_{\min} = \operatorname{argmin}_{e \in Y_{d,v} \setminus \{e^1_{\min}\}} w(e)$. (If $|Y_{d,v}| = 1$, set $e^2_{\min} = e^1_{\min}$.) Select the pair (d, v) that corresponds to the maximum difference $w(e^2_{\min}) - w(e^1_{\min})$ and add the vector e^1_{\min} for the selected (d, v) to the solution A .

In computational experiments, Balas and Saltzman [2] compared the greedy algorithm with max-regret and concluded that max-regret is superior to the greedy algorithm with respect to the quality of solutions. However, after conducting wider computational experiments, Robertson [25] came to a different conclusion: the greedy algorithm and max-regret are of similar quality for 3-AP. Gutin, Goldengorin and Huang [11] share the conclusion of Robertson: both greedy algorithm and max-regret are of domination number 1 for s -AP for each $s \geq 3$. Moreover, there exists a common family of s -AP instances for which both heuristics find the unique worst assignment [11] (for each $s \geq 3$).

Similarly to TSP, we may obtain MAP heuristics of factorial domination number if we consider not-worth-than-average heuristics. This follows from the next theorem:

Theorem 10. [11] *Let \mathcal{H} be a heuristic that for each instance of s -AP constructs an assignment of weight at most the average weight of an assignment. Then the domination number of \mathcal{H} is at least $((n - 1)!)^{s-1}$.*

Using Theorem 10, it is proved in [11] that the following heuristic is of domination number at least $((n - 1)!)^{s-1}$. The *Recursive Opt Matching* (ROM) heuristic proceeds as follows. Initialize the solution by trivial vectors: $e^i = (i, i, \dots, i)$, $i = 1, 2, \dots, n$. On each j th iteration of the heuristic, $j = 1, 2, \dots, s - 1$, calculate an $n \times n$ matrix $M_{i,v} = \sum_{e \in Y(j,i,v)} w(e)$, where $Y(j, i, v)$ is a set of all vectors $e \in X$ such that the first j coordinates of the vector e are equal to the first j coordinates of the vector e^i and the $(j + 1)$ th coordinate of e is v : $Y(j, i, v) = \{e \in X : e_k = e^i_k, 1 \leq k \leq j, \text{ and } e_{j+1} = v\}$. Let permutation π be a solution of the 2-AP for

the matrix M . We set $e_{j+1}^i = \pi(i)$ for each $1 \leq i \leq n$.

The *Multi-Dimensionwise Variation* (MDV) heuristic is introduced in [12] as a local search heuristic for MAP. MDV starts from the trivial solution $e^i = (i, i, \dots, i)$, $i = 1, 2, \dots, n$. On each step it selects a set of distinct dimensions, $F = \{f_1, f_2, \dots, f_k\}$, where $1 \leq k < s$. The corresponding dimensions are fixed, while the others are varied, and an $n \times n$ matrix $M_{i,j} = w(v^{i,j})$ is produced, where

$$v_d^{i,j} = \begin{cases} e_d^i & \text{if } d \in F \\ e_d^{(j)} & \text{if } d \notin F \end{cases} \quad \text{for } d = 1, 2, \dots, s.$$

Let permutation ρ be a solution of the corresponding 2-AP. If ρ is not an identity permutation, the heuristic changes the s -AP assignment in the following way:

$$e_d^i = \begin{cases} e_d^i & \text{if } d \in F \\ e_d^{(\rho(i))} & \text{if } d \notin F \end{cases} \quad \text{for } i = 1, 2, \dots, n \text{ and } d = 1, 2, \dots, s.$$

There are $2^s - 2$ distinct sets of the fixed dimensions F , but a half of them may be omitted since there is no difference whether to fix the selected dimensions and to vary the others, or to vary the selected dimensions and to fix the others. So, every iteration of the heuristic tries $2^{s-1} - 1$ distinct sets F . If no improvement was obtained during an iteration, the algorithm terminates. We have the following:

Theorem 11. *The domination number of MDV equals $(2^{s-1} - 1)(n! - 1) + 1$.*

Proof: Let a vector $e^i = (i, i, \dots, i) \in X$ for every $i = 1, 2, \dots, n$ and let vectors $e^{(i,j,F)} \in X$, $1 \leq i \neq j \leq n$, be defined as $e_k^{(i,j,F)} \in \{i, j\}$ for every $k = 1, 2, \dots, s$ and $e_k^{(i,j,F)} = i$ if and only if $k \in F$. We assign the weights as follows: $w(e^i) = 1$ for every $i = 1, 2, \dots, n$, $w(e^{(i,j,F)}) = 2$ for each of the $2^{s-1} - 1$ sets F and $1 \leq i \neq j \leq n$ and $w(e) = 0$ for every vector $e \in X$ that has at least three coordinates of different value. Let F_0 be the first set F chosen by MDV. Observe that for F_0 MDV outputs the trivial assignment e^1, \dots, e^n , which is the best among $n!$ assignments.

For every other F MDV outputs the trivial assignment which is better than $n! - 1$ assignments. Further iteration will output the trivial assignment as well. Thus, we conclude that the trivial assignment is the best among at most $(2^{s-1} - 1)(n! - 1) + 1$ assignments considered by MDV and, hence, the domination number of MDV is at most $(2^{s-1} - 1)(n! - 1) + 1$.

Now consider the last iteration of MDV. No improvement is made, and, thus, a solution with which we started the iteration will not change during the iteration. By permuting the elements of X_2, X_3, \dots, X_s (recall that $X = X_1 \times X_2 \times \dots \times X_s$), if needed, we may assume, without loss of generality, that the solution at the start of the last permutation is the trivial assignment. Since $e^{(i,j,F')} \neq e^{(i,j,F'')}$ provided $F' \neq F''$ and $F' \neq \{1, 2, \dots, s\} \setminus F''$, as above, we can see that the trivial assignment is the best among exactly $(2^{s-1} - 1)(n! - 1) + 1$ distinct assignments of the last iteration. Thus, $(2^{s-1} - 1)(n! - 1) + 1$ is a lower bound on the domination number of MDV. Since this lower bound is also an upper bound on the domination number, we are done. \square

This theorem and the result just after Theorem 10 show that ROM is of larger domination number than MDV for every fixed $s \geq 3$ for every n large enough. This is in contrast with the experimental results reported in Section 6, where the solutions obtained by MDV are almost always better than those produced by ROM. There is no contradiction in the two comparisons as they measure different sides of the quality of the two heuristics: the worst case behavior vs. the performance on some particular families of MAP instances.

5 Empirical Evaluation of Greedy Like Algorithms for TSP

We considered three ATSP heuristics in our experiments: Greedy, NN, and Patch Cycles (Patch) [22].

The Greedy heuristic is implemented as follows. Construct an array of all arcs $x_i y_i$, $x_i \neq y_i$, $1 \leq i \leq n(n-1)$, and sort this array by the arc weight: $w(x_i y_i) \leq w(x_{i+1} y_{i+1})$ for every $1 \leq i < n(n-1)$. Let $prev(i)$ be the vertex preceding the vertex i in the tour, and let $next(i)$ be the vertex succeeding the vertex i in the tour. Initialize $prev(i) = next(i) = 0$ for every $1 \leq i \leq n$. While the solution is incomplete, it consists of several separate components. For a vertex i let $id(i)$ be the identifier of the vertex i component. Initialize $id(i) = i$ for every $1 \leq i \leq n$. On every k th step of the heuristic try to add the arc $x_k y_k$ to the current solution, i.e., check whether $next(x_k) = 0$ and $prev(y_k) = 0$ and $id(x_k) \neq id(y_k)$. If all the conditions are met, set $next(x_k) = y_k$, $prev(y_k) = x_k$, and $id(i) = id(y_k)$ for every $i \in \{j : id(j) = id(x_k)\}$. When $n-1$ arcs are added to the solution, the

algorithm closes the cycle and stops.

The details on the NN heuristic are available in Section 3.

The `Patch` heuristic proceeds as follows. Let π be a solution of the assignment problem (AP) for the distance matrix of ATSP. Construct vertex-disjoint cycles c^i based on the AP solution π such that $c_{j+1}^i = \pi(c_j^i)$ for every $1 \leq j < s_i$ and $c_1^i = \pi(c_{s_i}^i)$, where s_i is the number of vertices in the i th cycle. Let m be the number of cycles, such that $\sum_{i=1}^m s_i = n$. If $m = 1$, the cycle c^1 is the optimal solution of ATSP and no further actions are required. Otherwise select two longest cycles (i.e., the cycles with the maximum values of s_i) and patch them by removing edges x_1x_2 from the first of them and y_1y_2 from the second one such that the value $w(x_1y_2) + w(y_1x_2) - w(x_1x_2) - w(y_1y_2)$ is minimized. Repeat this procedure until there is just one cycle, that is considered as a solution.

All the heuristics in this section and in Section 6 are implemented in Visual C++. The evaluation platform is based on AMD Athlon 64 X2 3.0 GHz processor.

The experiment results are reported in Tables 1 and 2. Table 1 includes the results for randomly generated instances of nine classes (for details see [13]). Ten instances of size 100, ten instances of size 316, three instances of size 1000, and one instance of size 3162 are considered for every instance class. The solution quality is presented by percent above the Held-Karp (HK) lower bound [20, 21].

Table 2 includes the results for several real-world ATSP instances from TSPLIB [24] and some other sources [20]. The solution quality is presented by percent above the best known solutions.

One can see that `Patch` clearly outperforms both `Greedy` and `NN` with respect to the solution quality, and the `NN` solutions are usually better than the `Greedy` ones (though `Greedy` slightly outperform `NN` on average with respect to the solution quality for the real-world instances). `NN` is much faster than both `Greedy` and `Patch`, while `Patch` is faster than `Greedy` for small instances and slower for the large ones. Johnson et al. [20] showed that, along with `Patch`, there are some other ATSP heuristics that are relatively fast and normally produce solutions that are much better than those obtained by `Greedy` and `NN`. (Some ATSP heuristics of good quality are also studied in [8].) Thus, it appears that `Greedy` should never be used in practice and `NN` is of interest only if a very fast heuristic is required.

6 Empirical Evaluation of Greedy Like Algorithms for MAP

In this chapter we consider four MAP heuristics: *Greedy*, First Coordinate Fixing (FCF), Recursive Opt Matching (ROM), and Multi-Dimensionwise Variation (MDV).

Greedy proceeds as follows. Let $A = \emptyset$ be a partial assignment and B an array of vectors. While $|A| < n$, i.e., A is not a full assignment, the following steps are repeated. Scan the weight matrix to fill array B with k vectors corresponding to k minimal weights and sort B in non-decreasing order. For each vector $e \in B$, starting from the lightest, check whether $A \cup \{e\}$ is a feasible partial assignment and, if so, add e to A . Note, that during the second and further cycles we scan not the whole weight matrix but only a subset $X' \subset X$ of the vectors that can be included into the partial assignment A with the feasibility preservation: $A \cup \{x\}$ is a partial assignment for any $x \in X'$. The size of the array B is calculated as $k = \min\{128, |X'|\}$.

The details on FCF, ROM and MDV heuristics are available in Section 4. As in [12], the number of MDV iterations was artificially restricted to 10.

The testbed includes three instance families: *Random*, *Composite*, and *GP*, discussed in [12].

In *Random Instance Family* (*Random*) the weight assigned to a vector is a random uniformly distributed integer value in the interval $[a, b - 1]$. We set $a = 1$ and $b = 101$. It is proved (see [12]) that the optimal solutions of large *Random* instances are very likely to be of weight an , so we assume in our experiments that the optimal solutions of the considered *Random* instances are exactly n .

The *Composite Instance Family* (*Composite*) is a family of semi-random instances. They were introduced by Crama and Spieksma for 3-AP as a problem T [7]. We extend this family for s -AP.

Let d^1, d^2, \dots, d^s be $n \times n$ matrices of non-negative uniformly distributed random integers in the interval $[a, b - 1]$. Let us consider a graph $G(X_1 \cup X_2 \cup \dots \cup X_s, (X_1 \times X_2) \cup (X_2 \times X_3) \cup \dots \cup (X_{s-1} \times X_s) \cup (X_1 \times X_s))$, where the weight of an edge $(i, j) \in X_k \times X_{k+1}$ is $d_{i,j}^k$ for $1 \leq k < s$ and the weight of an edge $(i, j) \in X_1 \times X_s$ is $d_{i,j}^s$. In this interpretation of s -AP, the objective is to find a set of n vertex-disjoint s -cycles $C \subset X_1 \times X_2 \times \dots \times X_s$ such that the total weight of

all edges covered by the cycles C is minimized.

In other words, $w(e) = d_{e_1, e_2}^1 + d_{e_2, e_3}^2 + \dots + d_{e_{s-1}, e_s}^{s-1} + d_{e_1, e_s}^s$.

The *GP Instance Family* (GP) contains pseudo-random instances with predefined optimal solutions. GP instances are generated by an algorithm given by Grunzel and Pardalos [10]. The generator is naturally designed for s -AP for arbitrary large values of s and n . The GP generator is relatively slow and, thus, it was impossible to experiment with large GP instances.

The results of the experiments are reported in Tables 3, 4, and 5. One can see that Greedy is significantly slower than the FCF heuristic, while its solution quality is not significantly better than the FCF's one. ROM outperforms or has very close results to Greedy with respect to both the solution quality and the running times. MDV clearly outperforms all other heuristics with respect to the solution quality, and it is the fastest algorithm for Random and Composite instances. For GP instances FCF and ROM are faster than MDV. Based on the experimental data, MDV is definitely the overall winner.

References

- [1] D.L. Applegate, R.E. Bixby, V. Chvátal and W.J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, 2006.
- [2] E. Balas, and M.J. Saltzman, An algorithm for the three-index assignment problem, *Operations Research* **39** (1991), 150–161.
- [3] J. Bang-Jensen, G. Gutin and A. Yeo, When the greedy algorithm fails, *Discrete Optimization* **1** (2004), 121–127.
- [4] H. Bekker, E.P. Braad and B. Goldengorin, Using bipartite and multidimensional matchings to select roots of a system of polynomial equations. In Proc. ICCSA'05, Lecture Notes in Computer Science **3483** (2005), 397–406.
- [5] G. Bendall and F. Margot, Greedy Type Resistance of Combinatorial Problems, *Discrete Optimization* **3** (2006), 288–298.
- [6] R.E. Burkard and E. Çela, Linear assignment problems and extensions, in *Handbook of Combinatorial Optimization*, Kluwer, Dordrecht, 1999, (Z. Du and P. Pardalos, eds.), 75–149.

- [7] Y. Crama and F.C.R. Spieksma, Approximation algorithms for three-dimensional assignment problems with triangle inequalities, *Europ. J. Operational Res.* **60** (1992), 273–279.
- [8] D. Ghosh, B. Goldengorin, G. Gutin and G. Jäger, Tolerance-based greedy algorithms for the traveling salesman problem, *Communications in DQM* **10** (2007), 52–70.
- [9] F. Glover, G. Gutin, A. Yeo and A. Zverovich, Construction heuristics for the asymmetric TSP, *European Journal of Operational Research* **129** (2001), 555–568.
- [10] D.A. Grundel and P. M. Pardalos, Test problem generator for the multidimensional assignment problem, *Comput. Optim. Appl.*, 30(2):133146, 2005.
- [11] G. Gutin, B. Goldengorin, and J. Huang, ‘Worst Case Analysis of Max-Regret, Greedy and Other Heuristics for Multidimensional Assignment and Traveling Salesman Problems’, *Lect. Notes Computer Sci.*, **4368** (2006), 214–225.
- [12] G. Gutin and D. Karapetyan, Local Search Heuristics For The Multidimensional Assignment Problem, Preprint arXiv:0806.3258v2.
- [13] G. Gutin and A.P. Punnen (eds.), *The Traveling Salesman Problem and its Variations*, Kluwer, 2002 and Springer-Verlag, 2007.
- [14] G. Gutin, A. Vainshtein and A. Yeo, When greedy-type algorithms fail, unpublished manuscript, 2002.
- [15] G. Gutin and A. Yeo, Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number, *Discrete Appl. Math.* **119** (2002), 107–116.
- [16] G. Gutin and A. Yeo, Anti-matroids, *Oper. Res. Lett.* **30** (2002), 97–99.
- [17] G. Gutin and A. Yeo, Domination Analysis of Combinatorial Optimization Algorithms and Problems. *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications* (M.C. Golumbic and I.B.-A. Hartman, eds.), Springer-Verlag, 2005.
- [18] G. Gutin and A. Yeo, The Greedy Algorithm for the Symmetric TSP. *Algorithmic Oper. Res.* **2** (2007), 33–36.

- [19] G. Gutin, A. Yeo and A. Zverovitch, Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP, *Discrete Appl. Math.* **117** (2002), 81–86.
- [20] D.S. Johnson, G. Gutin, L.A. McGeoch, A. Yeo, X. Zhang and A. Zverovitch, Experimental Analysis of Heuristics for ATSP, Chapter 10 in [13].
- [21] D.S. Johnson and L.A. McGeoch, Experimental Analysis of Heuristics for STSP, Chapter 9 in [13].
- [22] R.M. Karp, A patching algorithm for the non-symmetric traveling salesman problem, *SIAM J. Comput.*, 8:561573, 1979.
- [23] A.P. Punnen, F. Margot and S.N. Kabadi, TSP heuristics: domination analysis and complexity, *Algorithmica* **35** (2003), 111–127.
- [24] G. Reinelt, TSPLIB—A traveling salesman problem library, *ORSA J. Comput.* **3** (1991), 376-384, <http://www.crpc.rice.edu/softlib/tsplib/>.
- [25] A.J. Robertson, A set of greedy randomized adaptive local search procedure implementations for the multidimensional assignment problem. *Computational Optimization and Applications* **19** (2001), 145–164.
- [26] V.I. Rublineckii, Estimates of the Accuracy of Procedures in the Traveling Salesman Problem, *Numerical Mathematics and Computer Technology* no. 4 (1979), 18–23 [in Russian].
- [27] V.I. Sarvanov, The mean value of the functional of the assignment problem, *Vestsi Akad. Navuk BSSR Ser. Fiz. -Mat. Navuk* no. 2 (1976), 111–114 [in Russian].

Table 1: ATSP heuristics experiment results for randomly generated instances.

Family	n	Percents above HK			Running times, ms		
		Greedy	NN	Patch	Greedy	NN	Patch
amat	100	176.4	203.7	11.1	2.03	0.04	0.28
	316	246.1	231.3	6.5	25.97	0.45	2.91
	1000	283.4	337.4	2.7	319.18	4.38	50.34
	3162	344.1	397.3	1.9	3364.30	44.60	816.12
coin	100	26.9	27.7	16.0	1.67	0.05	0.16
	316	28.9	25.5	18.6	18.37	0.54	1.41
	1000	26.9	25.2	17.4	208.20	4.30	34.18
	3162	26.8	25.6	19.0	2382.02	46.26	431.35
crane	100	30.1	43.0	9.3	2.12	0.05	0.38
	316	45.3	60.0	13.6	26.03	0.46	8.92
	1000	66.1	81.0	31.1	338.89	5.10	156.46
	3162	105.9	113.2	58.4	3357.79	43.86	2366.64
disk	100	99.2	90.2	6.7	1.98	0.04	0.21
	316	148.7	107.7	2.2	26.37	0.40	5.72
	1000	296.1	117.3	0.9	321.94	4.35	205.25
	3162	566.5	162.0	0.3	3692.79	44.31	6935.62
rtilt	100	318.5	28.5	17.9	2.57	0.04	0.70
	316	651.6	28.2	17.9	26.59	0.41	10.06
	1000	1219.4	27.9	19.5	391.19	5.46	259.75
	3162	2260.9	24.2	21.3	3537.03	46.79	4477.49
shop	100	48.1	16.7	1.2	2.03	0.04	1.37
	316	55.9	14.8	0.6	20.90	0.43	26.89
	1000	60.4	13.5	0.4	209.73	4.61	939.74
	3162	63.8	11.9	0.2	2236.86	44.69	26662.96
stilt	100	103.8	32.6	25.3	2.08	0.05	0.88
	316	141.4	29.1	23.4	26.09	0.39	13.82
	1000	166.3	27.2	25.2	315.08	5.03	156.53
	3162	172.0	24.1	23.4	3428.08	44.22	2094.31
super	100	3.9	8.0	1.8	0.65	0.04	0.17
	316	4.1	8.8	2.7	6.70	0.94	2.55
	1000	4.5	9.9	4.2	73.66	4.75	70.06
	3162	4.9	10.1	6.1	815.01	61.75	959.90
tmat	100	26.3	38.0	0.9	2.11	0.04	0.21
	316	25.7	37.1	0.7	24.77	0.43	4.11
	1000	24.7	37.8	0.2	220.47	4.23	75.23
	3162	25.2	35.4	0.0	2041.57	45.73	1266.14
Avg.	100	92.6	54.3	10.0	1.92	0.04	0.48
	316	149.8	60.3	9.6	22.42	0.50	8.49
	1000	238.7	75.2	11.3	266.48	4.69	216.39
	3162	396.7	89.3	14.5	2761.72	46.91	5112.28

Table 2: ATSP heuristics experiment results for real-world instances. Here ∞ stands for ' $> 10^5$ ' and BK for 'best known.'

Instance	n	Percents above BK			Running times, ms		
		Greedy	NN	Patch	Greedy	NN	Patch
atex8	600	32.2	20.8	35.1	0.066	0.001	0.006
big702	702	17.9	35.6	0.0	0.090	0.002	0.028
code198	198	∞	∞	14.0	0.002	0.000	0.001
code253	253	59.0	∞	3.2	0.003	0.000	0.001
dc112	112	1.3	1.6	0.2	0.001	0.000	0.001
dc126	126	2.9	4.0	0.8	0.004	0.000	0.001
dc134	134	0.9	1.7	0.0	0.002	0.000	0.001
dc176	176	2.2	2.7	0.7	0.003	0.000	0.001
dc188	188	1.4	2.3	0.1	0.003	0.000	0.001
dc563	563	1.6	2.6	0.3	0.042	0.002	0.036
dc849	849	0.3	0.7	0.0	0.050	0.003	0.143
dc895	895	1.3	2.2	0.2	0.115	0.004	0.062
dc932	932	0.9	1.4	0.2	0.105	0.003	0.061
ftv100	101	60.6	44.1	1.9	0.002	0.000	0.000
ftv110	111	38.1	47.2	2.5	0.002	0.000	0.000
ftv120	121	45.3	38.8	6.6	0.002	0.000	0.000
ftv130	131	40.2	32.9	3.8	0.002	0.000	0.000
ftv140	141	38.1	42.2	3.5	0.003	0.000	0.000
ftv150	151	43.2	36.4	2.6	0.003	0.000	0.000
ftv160	161	50.4	51.5	1.7	0.003	0.000	0.000
ftv170	171	42.8	42.4	2.0	0.004	0.000	0.001
kro124p	100	21.0	31.1	13.8	0.002	0.000	0.000
rbg323	323	7.9	30.8	0.0	0.011	0.000	0.007
rbg358	358	8.0	55.8	0.0	0.012	0.000	0.009
rbg403	403	0.8	43.4	0.0	0.014	0.001	0.010
rbg443	443	0.7	44.2	0.0	0.021	0.001	0.012
Avg.		20.8	25.7	3.6	0.022	0.001	0.015

Table 3: MAP heuristics experiment results for Random instances.

s	n	Percents above optimal				Running times, ms			
		Greedy	FCF	ROM	MDV	Greedy	FCF	ROM	MDV
3	100	101.0	101.0	67.3	12.9	16.8	3.3	8.3	8.0
3	200	42.0	50.5	15.5	0.1	112.9	29.2	69.8	29.6
3	300	27.2	37.6	3.8	0.0	324.0	91.4	221.8	51.3
4	50	105.2	181.0	128.2	34.2	72.9	18.4	49.4	4.9
4	80	74.0	64.9	76.4	8.0	426.8	114.0	316.1	20.0
5	40	159.3	139.5	168.8	29.5	1072.4	232.2	803.5	8.7
6	20	277.5	302.5	327.0	41.0	693.8	156.1	509.1	3.8
7	12	420.8	425.0	475.0	57.5	412.5	84.6	294.5	2.7
8	8	548.8	550.0	723.8	62.5	207.2	44.3	152.2	2.2
Avg.		195.1	205.8	220.6	27.3	371.02	85.95	269.41	14.57

Table 4: MAP heuristics experiment results for Composite instances.

s	n	Percents above best known				Running times, ms			
		Greedy	FCF	ROM	MDV	Greedy	FCF	ROM	MDV
3	100	21.3	38.1	15.9	0.0	15.9	3.6	8.9	12.0
3	200	19.3	36.9	18.6	0.0	158.8	29.2	67.2	65.8
3	300	8.5	22.8	14.6	0.0	644.3	92.5	221.7	160.3
4	50	32.3	39.8	12.4	0.0	83.3	19.1	50.0	7.9
4	80	23.8	36.7	7.8	0.0	588.1	118.8	312.7	27.9
5	40	28.9	42.5	4.3	0.0	1350.9	241.6	932.5	11.4
6	20	35.3	43.4	3.8	0.0	812.9	142.1	563.6	5.5
7	12	36.0	43.8	5.7	0.0	504.1	85.0	294.8	4.0
8	8	28.5	34.9	7.3	0.0	233.9	45.4	146.5	3.1
Avg.		26.0	37.7	10.1	0.0	488.03	86.36	288.65	33.11

Table 5: MAP heuristics experiment results for GP instances.

s	n	Percents above optimal				Running times, ms			
		Greedy	FCF	ROM	MDV	Greedy	FCF	ROM	MDV
3	50	9.9	11.7	12.2	6.8	4.4	0.6	1.5	3.5
3	80	7.6	8.2	11.6	4.5	22.0	2.0	4.9	13.3
3	100	5.6	7.5	9.8	3.7	52.4	4.0	9.7	28.8
4	20	11.7	12.2	5.3	6.8	7.0	0.7	1.6	0.9
4	30	8.8	9.2	2.2	4.9	49.1	2.8	7.0	2.1
5	8	28.7	25.9	17.6	8.8	1.2	0.2	0.3	0.4
5	12	12.8	17.1	9.2	4.8	9.1	1.2	2.6	0.6
6	8	24.6	20.1	13.6	3.3	6.5	0.9	2.6	0.7
7	5	27.0	27.3	19.5	5.1	1.7	0.4	0.8	0.6
8	4	21.4	30.2	28.1	5.2	1.9	0.3	0.8	0.7
Avg.		15.8	16.9	12.9	5.4	15.5	1.3	3.2	5.2