Diversity as an Objective in Informational Retrieval
Experiments with a Navigation System

by
Jesse Funaro
Sc.B., Brown University 2003

A thesis submitted in partial fulfillment of the
requirements for the Degree of Master of Science
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2005

Diversity as an Objective in Informational Retrieval
Experiments with a Navigation System

by
Jesse Funaro
Sc.B., Brown University 2003

A thesis submitted in partial fulfillment of the
requirements for the Degree of Master of Science
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2005

Diversity as an Objective in Informational Retrieval
Experiments with a Navigation System

by
Jesse Funaro
Sc.B., Brown University 2003

A thesis submitted in partial fulfillment of the
requirements for the Degree of Master of Science
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2005

This thesis by Jesse Funaro is accepted in its present form by
the Department of Computer Science as satisfying
the thesis requirement for the degree of Master of Science.

Date ______________ ____

Amy Greenwald, Advisor

Approved by the Graduate Council

Date ______________ ____

Karen Newman
Dean of the Graduate School

This thesis by Jesse Funaro is accepted in its present form by
the Department of Computer Science as satisfying
the thesis requirement for the degree of Master of Science.

Date ______________ ______________________________
Amy Greenwald, Advisor

Approved by the Graduate Council

Date ______________ ______________________________
Karen Newman
Dean of the Graduate School

Vita

Jesse Funaro was born on November 17, 1980 in Camden, NJ. He lived his entire childhood in Moorestown, NJ, attending Moorestown High School before coming to Brown University in 1999. He obtained an Sc.B in Computer Science in 2003.

Acknowledgements

I would like to thank my advisor, Professor Amy Greenwald, for her guidance and understanding during my development of this thesis. The introductory problem descriptions at the beginning of the thesis are due to Professor Greenwald.

Contents

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

1.1 Contextual Preferences 2

1.2 Selection Problem 3

1.2.1 Formal Problem Statement 3

1.3 Heuristic Solutions 5

2 Experiments with a Navigation System **7**

2.1 Data Set 7

2.2 Task Generation 7

2.3 Preference Representation 8

2.4 Population Generation 9

2.5 Candidate Solution Set Generation 9

2.6 Task "Difficulty" 10

2.7 Multi-A^* 11

2.8 Implementation of the Naive Algorithm 12

2.9 Implementation of the Greedy Heuristic Algorithm 12

2.10 Implementation of the Hill-Climbing Algorithm 17

3 Results **19**

3.1 Preliminary Results 19

3.1.1 Technical Details 21

3.2 Further Results 21

4 Related Work **35**

5 Future Work **36**

6 Conclusion **39**

Bibliography **40**

List of Tables

2.1 Correspondence between task length indicator and actual length 8
2.2 Number of unique routes obtained, r = 50, n = 50 10
2.3 Number of unique routes obtained, r = 100, n = 25 10
2.4 Mean versus Median set generation times 10

3.1 Mean & standard deviation of cost, 552 10-mile tasks. 19
3.2 Mean & standard deviation of cost, 512 15-mile tasks. 20
3.3 Mean & standard deviation of cost, 526 20-mile tasks. 20

List of Figures

2.1 Min-Priority Queue helper function for Multi-A* 12
2.2 Multi-A* Pseudocode 13
2.3 Proof of equivalence between first choices of naive and greedy algorithms 14
2.4 Helper functions for Greedy and Hill climbing algorithms 15
2.5 Greedy Selection Pseudocode 16
2.6 Hill Climbing Pseudocode 18

3.1 Algorithm performance with basis population on a 10-mile task 22
3.2 95% confidence intervals over algorithm performance with normally distributed, correlated population on a 10-mile task 23
3.3 95% confidence intervals over algorithm performance with normally distributed, independent population on a 10-mile task 24
3.4 95% confidence intervals over algorithm performance with uniformly distributed, correlated population on a 10-mile task 25
3.5 95% confidence intervals over algorithm performance with uniformly distributed, independent population on a 10-mile task 26
3.6 95% confidence intervals over algorithm performance with normally distributed, correlated population on a 15-mile task 27
3.7 95% confidence intervals over algorithm performance with normally distributed, independent population on a 15-mile task 28
3.8 95% confidence intervals over algorithm performance with uniformly distributed, correlated population on a 15-mile task 29
3.9 95% confidence intervals over algorithm performance with uniformly distributed, independent population on a 15-mile task 30
3.10 95% confidence intervals over algorithm performance with normally distributed, correlated population on a 20-mile task 31
3.11 95% confidence intervals over algorithm performance with normally distributed, independent population on a 20-mile task 32
3.12 95% confidence intervals over algorithm performance with uniformly distributed, correlated population on a 20-mile task 33

3.13 95% confidence intervals over algorithm performance with uniformly distributed, independent population on a 20-mile task 34

Chapter 1

Introduction

In the classic theory of expected utility (see, for example, [14]), an individual faces a decision among lotteries, or options with uncertain outcomes. For example, if an individual is driving from San Francisco to NASA Ames Research Center, he might ask himself, "should I take Highway 101 or Interstate 280?" In making this decision, the individual trades off distance against scenery against travel time, weighing in the uncertainty about traffic conditions, to estimate the utility of each option. According to this theory, the decision that is taken is one that maximizes expected utility.

In this paper, we study what we call the *meta-choice problem*, namely "Given a query, what set of options should an information retrieval system present to a user?" given that the user is an expected utility maximizer. Naive solutions to this problem may produce options that satisfy a typical user's wants and needs—for example, in the driving directions case, presenting a set of ten options comprising the ten quickest routes (in expectation). A set of such options, while maximizing the expected utility of a typical user, is likely to be highly redundant, and thus unsatisfactory across a diverse user population.

In formulating the meta-choice problem, we propose objectives we believe are more suitable than maximizing the expected utility of a typical user, and we describe practical algorithms to optimize these objectives. The output of such algorithms is a small set of diverse options, among which most users are likely to find a desirable choice. Some specific applications of meta-choice to information retrieval systems include the following:

- *best-*1 *selection problem*: This problem is faced by `orbitz.com`, for example, whose goal is to suggest several alternative travel itineraries to a user, even if the user will ultimately take only one.

- *best-k selection problem*: This problem is faced by `news.google.com`, for example, whose goal is to provide pointers to several articles, preferably coming from different perspectives, in response to a user's query about, say, "Saddam Hussein."

- Search engines: Given the query "python," return some links about snakes & some about the comedian, in addition to links about the programming language.

- Recommender systems: Given the query "laptop, under 3 pounds and under \$3,000," return some IBMs, some MACs, some Dells, etc.

1.1 Contextual Preferences

Our approach to the meta-choice problem depends on an adequate representation of user preferences. In multiattribute decision theory [6], preferences are described in terms of various attributes, or features, of the alternatives: e.g., in the driving example above, relevant attributes may include distance, scenery, travel time, number of turns, number of lights, and highway time. For some kinds of decisions, however, this form of representation is far too rigid. Indeed, context-dependent preferences [13] are an alternative to the classical theory of choice, which relies on the assumption of stable preferences.

We introduce an alternative context-based representation of user preferences. Formally, a context is characterized by n discrete attributes. In particular, attribute j takes values in the set Ω_j, and the set of all contexts is given by $\Omega = \prod_{j=1}^{n} \Omega_j$. We assume that the ith context occurs with probability p_i. These probabilities model the fact that sometimes people travel for business, while other times people travel for pleasure; and sometimes it is sunny, but other times it is raining; etc.

We rely on multiattribute utility functions to describe user preferences. These functions are defined on the set of options S, where each option $\vec{x} \in S$ is characterized by m features (or attributes). In particular, the utility function $U_i : S \to \mathbb{R}$ describes the user's preferences in the ith context. A simple example of a utility function is, for option $x \in S$, compute a linear combination of x's feature values: i.e., $U_i(x) = \vec{w}_i \cdot \vec{x} = \sum_{k=1}^{m} w_{ik} x_k$, for some weighting $\vec{w}_i$ of the features. Note that our contextual representation of preferences is nonlinear.

Our contextual representation of preferences should *not* be interpreted as a noisy representation of a user's true underlying preferences, in which case the user's optimal choice (or set of choices) could be computed w.r.t. the expected value of his contextual preferences. Rather, it is designed to model different contexts under which a user's optimal choices are truly different. An optimal solution to the meta-choice problem should be a set of options that satisfies the user in multiple contexts.

Our contextual representation of preferences is sufficiently general to be used to model either a single user, in which case each context describes attributes of that user and his environment, or an entire population of users (where each user is associated with one or many contexts). An information retrieval system could adopt the former point of view if it had knowledge of the user performing the search, or the latter if it had knowledge about the user population but not the individual user.

Given contextual preferences, one approach to meta-choice would be to probe the user for his precise context, and to respond to his query with his preferred option in that context. This explicit approach, however, would be invasive and potentially annoying to users. Alternatively, in this paper, we advocate the design of information retrieval systems that solve the meta-choice problem by optimizing w.r.t. uncertainty over context.

1.2 Selection Problem

We assume that an individual's (or population's) preferences can be constructed by combining context-dependent utility functions according to the likelihood of each context. Given this contextual representation of preferences, the objective is to present a set of options to the user in response to a query that maximizes his utility, *in expectation.* This expectation is computed with respect to the uncertainty over contexts. Recall that the context is known only to the user—it is too invasive for the system to query the user for his context.

We now formally describe one variety of meta-choice, namely the *selection* problem, and propose several algorithmic solutions. This setup is not specific to navigation systems, but is applicable to many information retrieval (IR) systems, including `orbitz.com` and `news.google.com`. We describe the problem and our algorithms with this more general framework in mind.

1.2.1 Formal Problem Statement

Assume the set S of size $L \gg 0$ contains all relevant options that might be selected by an IR system in response to a user's query. The objective of an IR system is to select some subset $T \subseteq S$ of options to present to the user. Typically, $|T| = l \ll L$. Given this set of options, the user then chooses $T' \subset T$, with $|T'| \leq l$. Before we can define the IR system's objective function, we must first make some assumptions about how the user makes his choices. Ultimately, the IR system's objective depends on the user's behavior, which in turn depends on his (contextual) preferences.

We assume the user's preferences are described by a context-based distribution over multiattribute utility functions. In particular, we are given a probability distribution p over contexts, where p_i denotes the probability of the ith context. Corresponding to each context is a *combinatorial*, multiattribute utility function $C_i : 2^S \to \mathbb{R}$, which assigns utility to subsets of options.

Our formulation of the selection problem is based on the assumption that the user is a utility maximizer: Given the IR system's selection $T \subseteq S$, a utility maximizing user optimizes his combinatorial utility function over all subsets $T' \subseteq T$ of size no more than k.

User Behavior: Given selection $T \subseteq S$, we assume the user maximizes his combinatorial, multiattribute utility function C_i in the ith context as follows:

$$\max_{\substack{T' \subseteq T \\ |T'| \leq k}} \quad C_i(T')$$

Depending on the *task* at hand (not the context), the assumption that the user is utility maximizing can give rise to a variety of behaviors. For example, in some tasks, such as news analysis, the user reads a small (say k) number of articles; in other tasks, such as transportation assistance, the user picks a single route ($k = 1$).

Given this assumption about how the user solves his optimization problem, we now state the selection problem, which is the optimization problem faced by the IR system. If the current context is known, say i^*, then the IR system aims to solve the following context-dependent optimization problem:

Context-Dependent Selection: Given combinatorial, multiattribute utility function C_{i^*}, choose a subset $T \subset S$ of size no more than l that maximizes the user's utility: i.e., find

$$T^* \in \arg \max_{\substack{T \subseteq S \\ |T| \le l}} \max_{\substack{T' \subseteq T \\ |T'| \le k}} C_{i^*}(T')$$

In general, the context is unknown, and the IR system aims to solve the following stochastic version of the aforementioned optimization problem, which yields a context-independent solution. In this, our most general formulation of selection, the user is interested in at most k options among those presented by the system.

Best-k Selection: Given combinatorial, multiattribute utility functions C_i, choose a subset $T \subset S$ of size no more than l that maximizes the user's utility, *in expectation*: i.e., find

$$T^* \in \arg \max_{\substack{T \subseteq S \\ |T| \le l}} \sum_i p_i \left(\max_{\substack{T' \subseteq T \\ |T'| \le k}} C_i(T') \right)$$

The best-1 selection problem is the special case of best-k selection in which $k = 1$. In the ith context, the user's evaluation of a set T is determined by the quality of the best single option $x \in T$ according to utility function $U_i : S \to \mathbb{R}$: i.e., $C_i(T) = \max_{x \in T} U_i(x)$, given $T \subset S$. Such preferences are called *unit-demand preferences.* Best-1 selection specializes best-k selection:[1]

Best-1 Selection: Given multiattribute utility functions U_i, choose a subset $T \subset S$ of size no more than l that maximizes the user's utility, *in expectation*: i.e., find

$$T^* \in \arg \max_{\substack{T \subseteq S \\ |T| \le l}} \sum_i p_i \left(\max_{x \in T} U_i(x) \right)$$

The best-1 selection problem is an appropriate description of many practical IR applications, and the problem on which we focus our experiments.

[1]Note the following: for $T \subset S$, $\max_{T' \subseteq T} C_i(T') = \max_{T' \subseteq T} \max_{x \in T'} U_i(x) = \max_{x \in T} U_i(x)$.

1.3 Heuristic Solutions

One naive approach to approximating a solution to the selection problem is to average the distribution of utility functions over contexts, thereby arriving at a description of the user in the typical context, and to solve the context-dependent selection problem in this typical case. This so-called *expected value* method, which ignores most of the distributional information, is well-known to be suboptimal in general [2]. Several alternatives, which make further use of distributional information, are possible.

Hill-Climbing

Local search is an alternative approach by which to solve the selection problem. Initialize the search at random to a subset $T \subset S$ with $|T| = l$. Score this subset using $\sigma : 2^S \to \mathbb{R}$, as follows:

$$\sigma(T) = \sum_i p_i \left(\max_{\substack{T' \subseteq T \\ |T'| \leq k}} C_i(T') \right)$$

As an improvement step, evaluate all the elements of the current solution, and replace, say, the worst element (given some means of scoring options) with another, particularly if an improvement is found.

Time permitting, random restarts can be used with such a search procedure to escape local optima.

One means of scoring options $x \in S$ is via marginal utilities. The marginal utility of an option $x \in S$ relative to a set T is the added benefit obtained by x's presence. Given combinatorial utility function $C_i : 2^S \to \mathbb{R}$ in context i, for all $x \in S, T \subseteq S$, the marginal utility $\mu_i(x \mid T)$ of option $x \in T$ relative to set T is given by:

$$\mu_i(x \mid T) = C_i(T) - C_i(T \setminus \{x\}) \tag{1.1}$$

The expected value of the marginal utility is defined in the obvious way:

$$\overline{\mu}(x \mid T) = \sum_i p_i \mu_i(x \mid T) \tag{1.2}$$

A Greedy Heuristic

The following greedy heuristic is applicable to the context-dependent selection problem: starting with the empty set, add options in non-increasing order of marginal utility, until the set contains l options. This greedy heuristic is a $(1-1/e)$-approximation algorithm of the NP-hard problem: choose a subset $T \subset S$ of size no more than l that maximizes the *submodular* function f, assuming free disposal: i.e., $f(T_1) \leq f(T_2)$ whenever $T_1 \subseteq T_2$ [10, 11]. Submodularity, which captures the idea of diminishing returns, can be defined in terms of marginal utility. A function $f : 2^S \to \mathbb{R}$ is submodular iff $T_1 \subseteq T_2 \subseteq S$ implies $\mu_f(x \mid T_1) = f(T_1) - f(T_1 \setminus \{x\}) \geq f(T_2) - f(T_2 \setminus \{x\}) = \mu_f(x \mid T_2)$. The submodularity property implies that the marginal benefit of adding x to the set of options T_1 is at least as great as the marginal benefit of adding x to the set of options T_2 whenever $T_1 \subseteq T_2$.

To solve, for example, the selection problem, we propose the following variant of the aforementioned greedy heuristic: starting with the empty set, add options in non-increasing order of the *expected value* of marginal utility, until the set contains l options.

Theorem This greedy heuristic is a $(1-1/e)$-approximation algorithm of the NP-hard problem: choose a subset $T \subset S$ of size no more than l that maximizes the submodular function $g(T) = \sum_i p_i f_i(T)$, given $f_i(T)$ submodular for all i and $\sum_i p_i = 1$, and assuming free disposal.

Proof A non-negative linear combination of submodular functions is again submodular. In particular, $g(T)$ is submodular. Moreover,

$$\begin{aligned}
\mu_g(x \mid T) &= g(T) - g(T \setminus \{x\}) && (1.3)\\
&= \sum_i p_i \left(f_i(T) - f_i(T \setminus \{x\})\right) && (1.4)\\
&= \sum_i p_i \mu_i(x \mid T) && (1.5)\\
&= \overline{\mu}(x \mid T) && (1.6)
\end{aligned}$$

Now the result follows immediately via the theorem in [10, 11].

Observation The function $C_i(T) = \max_{x \in T} U_i(x)$ is submodular, assuming free disposal.

Proof Assume $T_1 \subseteq T_2$. We must show $\mu(x \mid T_1) \geq \mu(x \mid T_2)$. Equivalently, we must show $C_i(T_1) - C_i(T_1 \setminus \{x\}) \geq C_i(T_2) - C_i(T_2 \setminus \{x\})$. Three cases arise:

- $C_i(T_1) > C_i(T_1 \setminus \{x\})$ and $C_i(T_2) > C_i(T_2 \setminus \{x\})$: In this case, $C_i(T_1) = C_i(T_2) = U_i(x)$. This observation, together with the fact that $C_i(T_2) \geq C_i(T_1)$ (i.e., free disposal), implies the result.

- $C_i(T_1) > C_i(T_1 \setminus \{x\})$ and $C_i(T_2) = C_i(T_2 \setminus \{x\})$: $C_i(T_1) - C_i(T_1 \setminus \{x\}) > 0 = C_i(T_2) - C_i(T_2 \setminus \{x\})$.

- $C_i(T_1) = C_i(T_1 \setminus \{x\})$ and $C_i(T_2) = C_i(T_2 \setminus \{x\})$: $C_i(T_1) - C_i(T_1 \setminus \{x\}) = 0 = C_i(T_2) - C_i(T_2 \setminus \{x\})$.

The fourth case, namely $C_i(T_1) = C_i(T_1 \setminus \{x\})$ and $C_i(T_2) > C_i(T_2 \setminus \{x\})$ does not arise for this particular choice of combinatorial utility function.

Remark If we define $C_i(T)$ in terms of combinatorial utility functions: i.e.,

$$C_i(T) = \max_{T' \subseteq T} C_i(T')$$

this fourth case could indeed arise. Rule it out by assuming *no complementarities.*

Corollary Our greedy heuristic is a $(1 - 1/e)$-approximation algorithm for solving best-1 selection, assuming free disposal.

Chapter 2

Experiments with a Navigation System

In the remainder of this paper, we describe simulation experiments we conducted to solve the best-1 selection problem in an experimental navigation system. Recall that in best-1 selection problem, the user picks (at most) one of the options presented by the IR system, namely that which maximizes his utility. For example, a user searching for a route from Providence, RI to Cambridge, MA is shown several solutions to his query, although ultimately he takes only one route.

2.1 Data Set

Our experiments utilized data obtained from the Rhode Island Geographic Information System (RIGIS). The RIGIS data contains segmented representations of the roads in Rhode Island, specifying multiple characteristics for each, e.g., road grade, speed limit, length. The data also contains a series of planar point locations corresponding to each road segment. Intersections are defined by their planar point location and a list of connected streets. We augmented this data by randomly distributing fictitious stop signs and traffic lights, natural choices for route attributes that might also be important to a user.

2.2 Task Generation

We simulated a series of user queries for driving directions. In order to test our algorithms on a variety of routing queries, we generated routing tasks at random. A routing task is defined as a pair of intersections. First, an arbitrary source intersection was chosen. Next, the data set was probed at a random for an appropriate destination intersection. The first intersection found within a small threshold of the desired distance from the source was selected, yielding a complete routing task. Repeating this process allowed for fast and easy generation of large bodies of routing tasks.

It is important to note that distances between point locations in the data set correspond to real-world distances. That is to say, the Euclidean distance between two intersections in the data set is approximately equivalent to the actual "as the crow flies" distance between the intersections.

For our tests, we generated routing tasks within three broad categories, which we refer to as 10, 15, and 20 mile tasks, indicating the desired distances of the tasks in each category. Not surprisingly, the "as the crow flies" distance between two points is a crude indicator of the actual length of a path between them.

As an extreme example consider two points on opposite sides of the Strait of Gibraltar, approximately 13km wide at its narrowest. A path along streets between these two locations (if such a path exists) would almost certainly be much longer than 13km.

Nonetheless, we refer to tasks by their length indicator alone, e.g., "a 10 mile task", when, in fact, the (shortest path) solution to the task may not be of length exactly 10 miles. The approximate correspondence between our generation parameters and the actual lengths of the shortest solutions to the routing tasks over a series of trials is shown in Table 2.1.

Task Length Indicator	# Trials	Mean Length (mi.)	Std Deviation (mi.)
		μ	σ
10 mile	392	17.35	6.61
15 mile	396	24.64	9.31
20 mile	389	30.71	7.55

Table 2.1: Correspondence between task length indicator and actual length

2.3 Preference Representation

In these experiments, we relied on our contextual model of preferences, that is, we assume a population of users, each with his own individual preferences. To represent of an individual's preferences, we assume a weighted additive utility model. In particular, given vector x describing an option's attributes and user i's "profile" (i.e., weight vector) w_i, user i's utility function $U_i(x) = w_i \cdot x$.

Although we rely on a weighted additive utility model, the units with which our route attributes are measured cannot be added together to yield meaningful results. It is nonsensical to take linear combinations of, for example, distance in feet and number of stop signs. Our simple solution to this problem is to use normalized costs for each of the attributes, where a route is described not by its actual attributes, but by ratios of each of its attribute values to the best possible attribute value, given the routing task. We determine these "cost normalization factors" by computing the extremal solutions to the routing task: e.g., the solution optimizing for distance alone, or for time alone, or for number of road segments alone. Once the cost normalization factors for the task have been obtained, the cost to a user of a route is a weighted linear combination of that route's normalized attributes. Two of our attributes, number of stop signs and number of stop lights, have the possibility of taking on value zero for some routing tasks. In these cases we must make an arbitrary choice for the normalization factor, which we set at $\frac{1}{100}$. In this way, we penalize the use of a stop sign or light with cost 100 whenever an extremal solution with zero stop signs or lights exists.

2.4 Population Generation

A population is a set of profiles describing user preferences. Each user profile in a population is defined as a vector of length 5 with all entries between 0 and 1, inclusive. Each entry is a measure of the cost that a user assigns to the five attributes: distance, pass time, number of unique roads traveled, number of stop signs, and number of stop lights.

We considered 15 different populations. The first of these is what we refer to as the "basis population," the population containing users who assign cost 1 to a single characteristic and cost 0 to all others, with each basis user present in equal proportions. Next, there are the "planar" populations made up of two unique profiles chosen from the set of basis profiles, in equal proportions. There are $\binom{5}{2} = 10$ such planar populations.

We also considered 2 populations drawn from uniform random distributions. The first was a population for which each cost element in a profile was generated independently, and the second was a population with correlated costs for distance and pass time (correlation coefficient $\rho \approx .74$) Finally, we considered 2 populations drawn from normal distributions. The first was a population for which each cost in a profile was generated independently according to the normal distribution $N(0.5, 0.1)$. In the second, 3 elements were independently distributed according to the normal $N(0.5, 0.1)$, but costs for distance and pass time were correlated (correlation coefficient $\rho = 0.8$). Distance and pass time costs are drawn from a bivariate normal with mean 0.5 as above, and covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{bmatrix} = \begin{bmatrix} (.1)^2 & .008 \\ .008 & (.1)^2 \end{bmatrix}$$

This covariance matrix by construction yields a distribution with correlation coefficient $\rho = \frac{\sigma_{21}}{\sigma_1 \sigma_2} = 0.8$.

2.5 Candidate Solution Set Generation

Our algorithms rely on a relatively large candidate set of solutions from which to choose the "result set." In order to generate a candidate solution set for a routing task, we first computed the extremal solutions to the routing task, namely the optimal routes for each profile in the basis population. We then added the top m routes computed by the naive approach (described below) to this set. Finally, we added additional routes generated by repeated calls to a modified A^*-search algorithm, which we call multi-A^* (also described below), with randomly generated user profiles as input. We used multi-A^*, with an inadmissible heuristic, rather than Dijkstra's algorithm, which is optimal, in the interest of time.

Task	μ	σ	avg. time
10 mile	1084.81	594.1	42.12s
15 mile	1388.67	619.2	69.73s
20 mile	1534.51	567.8	75.25s

Table 2.2: Number of unique routes obtained, r = 50, n = 50

Task	μ	σ	avg. time
10 mile	798.96	490.4	24.90s
15 mile	1039.69	552.2	70.79s
20 mile	1110.90	568.4	117.99s

Table 2.3: Number of unique routes obtained, r = 100, n = 25

The size of the candidate solution set is controlled by two parameters: the number r of random vectors with which multi-A^* is called, and the number n of solutions requested from each run of multi-A^*.

Data from preliminary tests of 100 trials with $r = 50$, $n = 50$ are shown in Table 2.2, and data from 100 trials with $r = 100$, $n = 25$ are shown in Table 2.3.

Intuitively, a solution set generated with a larger value of r represents a greater variety of routes. For this reason, and since that we were easily generating sets of an adequate size for our purposes, we chose to generate solution sets for our experiments with parameters $r = 100$, $n = 25$.

We chose to invoke the multi-A^*algorithm with 100 user profiles, uniformly generated at random, as input, requesting 25 results each time. These parameters struck an appropriate balance of processing time, candidate solution set size, and variety for our purposes. Typical set sizes generated using this method were between 800 and 1500, with the 10 mile tasks tending to smaller set sizes and 15 and 20 mile tasks tending towards larger.

2.6 Task "Difficulty"

One characteristic of our search space made apparent by our experiments is a large disparity in the inherent "difficulty" of generating multiple solutions to a routing task. We observed a large variance in the processing time necessary to generate our sets. In the following table, we present the mean, median, and standard deviation of set generation times over roughly 520 trials :

Task	μ	median	σ
10 mile	149.41s	16.98s	478.7s
15 mile	235.74s	30.90s	971.9s
20 mile	317.82s	40.53s	1146.4s

Table 2.4: Mean versus Median set generation times

It is evident from the large standard deviation of generation times, as well as from the clear disparity between median and mean generation times, that a few tasks proved to be quite hard and took quite a long time, while the majority of our tasks were much less difficult. This is most likely due to high variation of the depth within the search space of paths necessary before the multi-A^*algorithm found viable solutions.

2.7 Multi-A^*

Our multi-A^* algorithm is a straightforward extension of A^*-search, designed to return multiple solutions. Whereas a typical A^*-search would terminate once a goal was reached, we append the completed path to our list of solutions, clear the list of closed nodes to allow backtracking, and continue to remove elements from the queue of unexplored nodes as if the goal had not yet been found. The multi-A^* algorithm terminates in one of two cases: either the queue has been exhausted, and thus no more solutions can be found, or the solution set is of the desired size. Our heuristic in multi-A^* estimates a ratio of the cost to the user per unit length of the route. We calculate the ratio of the total user cost incurred thus far to the total Euclidean distance covered thus far. This ratio multiplied by the Euclidean distance remaining to the destination is a rough estimate of the remaining cost to the user. Here, we assumed that roads available from the current point onward will match the profiles of roads traveled thus far. This is a reasonable assumption given that our routing tasks do not cover excessively large distances. Nevertheless, this heuristic is inadmissible.

It is important to discuss the process of choosing a heuristic measure for our multi-A^*. We do not present an admissible heuristic. Because we use multi-$A^*$with a set of randomly generated preference vectors to generate a diverse set of routing options, returning the optimal solution for any one of these vectors is not necessary. We would, nevertheless, like the multi-A^*solution to be close to optimal.

First, there are two points to consider. Multi-$A^*$with a zero heuristic is equivalent to what we refer to as "multi-Dijkstra", i.e. a modification of Dijkstra's algorithm to return multiple solutions. The second point to consider is that for the problem of shortest path alone, crow flies distance is an admissible heuristic and multi-$A^*$is optimal. In our specific problem formulation, however, the cost of a street arc is not its length alone, but a linear combination of 5 street characteristics. The crow flies distance is thus an inadmissible heuristic for our problem.

We present two other inadmissible heuristics which intuitively and experimentally outperform the crow flies distance.

The first heuristic utilizes local cost information, specifically the ratio of the cost of the previous street arc to its length. The resultant value is a measure of user cost per unit distance, which, when multiplied by the crow flies distance remaining to the destination, constitutes our heuristic value. When a particular user profile expresses preference for shortest path alone, user cost per unit distance is equal to one, and thus this heuristic maintains optimality for users whose only desire is

```
ADD(MinPriorityQ fringe, HashTable fringed, SearchElement i)
if $fringed(i.intersection) = \emptyset$
  then //insert element with priority f
        $fringe.add(i.f, i)$;
        $fringed.add(i)$;

  else
        if $fringed.get(i.intersection).f > i.f$
          then $fringed.remove(i.intersection)$;
                $fringe.add(i.f, i)$;
                $fringed.add(i)$;
```

Figure 2.1: Min-Priority Queue helper function for Multi-A*

shortest path. In general, however, this heuristic is inadmissible.

The second heuristic, much like the first, attempts to formulate a ratio of cost to the user per unit length, but has a more global outlook in its calculation. This heuristic uses the ratio of the total user cost thus far to the total crow flies distance traveled thus far. This ratio multiplied by the crow flies distance remaining to the destination is a heuristic which does not, as the first, preserve optimality for user desiring only shortest paths. This heuristic, however, with uniform random profiles, outperforms both previous heuristics, and was that which was eventually chosen.

2.8 Implementation of the Naive Algorithm

The naive approach chooses routes based on costs to the typical user. In particular, the set of l routes chosen by the naive algorithm includes the l shortest paths, with costs given by the expected user profile. The resulting set of routes can be obtained from a call to "multi-Dijkstra," (multi-A^* with a 0 heuristic), with the expected user profile and the desired result set size as input.

2.9 Implementation of the Greedy Heuristic Algorithm

The greedy heuristic adds routes to the result set in non-decreasing order of average marginal cost. Our implementation iteratively calculates the average marginal cost of each candidate element in the following manner: the cost of each route r is calculated across the entire population, for each user taking the minimum of the user's cost for r and the user's cost for his preferred route in the result set thus far. In the first iteration, this set is empty, and thus the first route chosen is identical to the first route found by the naive approach. (Proof, Figure 2.3) Beyond this first iteration, however, the greedy heuristic ignores segments of the population satisfied by routes in previous iterations, and thus selects a wider variety of routes than the naive approach.

```
MULTI-A*(Graph G, Isec S, Isec T, WeightVector V, int n)
$fringe$ :min-priority queue
$routes$ :empty list
$visited$ :hashtable of visited intersections
$fringed$ :hashtable of intersections currently on the fringe
$curr = \{S, 0, \|S - T\|, \emptyset\}$
while !$stop$
do
   while $curr.intersection \neq T$
   do
      if $visited(curr.intersection) = \emptyset$
        then
             $visited.add(curr.intersection)$;
             for intersections $i$ reachable from $curr.intersection$
             do
                if $visited(i) = \emptyset$
                  then
                       $g \leftarrow curr.cost + costOfStreet(V, curr.intersection, i)$;
                       $distanceSoFar \leftarrow \|S - i\|$;
                       $h \leftarrow \frac{g}{distanceSoFar} \cdot \|i - T\|$;
                       $f \leftarrow MAX(g + h, curr.f)$;
                       $newElement \leftarrow \{i, g, f, curr.isecList \circ i\}$;
                       ADD$(fringe, fringed, newElement)$;


             if $fringe.size = 0$
               then $stop \leftarrow true$;
                    $break$;

      $curr \leftarrow fringe.first()$;
      $fringed.remove(curr.intersection)$;

   if !$stop$
     then
          $routes.add(current.isecList)$;
          $visited.clear()$;
          if $routes.size() = n$
            then return $routes$;


   if $fringe.size = 0$
     then
          $stop \leftarrow true$;
          $break$;

     else $curr \leftarrow fringe.first()$;
          $fringed.remove(curr.intersection)$;

return $routes$
```

Figure 2.2: Multi-A* Pseudocode

Claim Assume an additive utility function. The average utility a population attributes to an option x is equal to the utility of the option x evaluated with respect to the average individual.
Proof

$$\begin{aligned}\overline{U}(x) &= \sum_i p_i U_i(x) && (2.1)\\ &= \sum_i p_i\left(w_i \cdot x\right) && (2.2)\\ &= \sum_i p_i \left(\sum_j w_{ij} x_j\right) && (2.3)\\ &= \sum_j x_j \left(\sum_i p_i w_{ij}\right) && (2.4)\\ &= \sum_j x_j \overline{w}_{ij} && (2.5)\\ &= x \cdot \overline{w}_i && (2.6)\end{aligned}$$

Corollary The solution to best-1 selection for $l = 1$ is the best option (i.e., that which yields highest utility) evaluated with respect to an average individual.
Proof

$$\begin{aligned}\arg\max_{T\subset S} \sum_i p_i \left(\max_{x\in T} U_i(x)\right) &= \arg\max_{x\in S} \sum_i p_i U_i(x)\\ &= \arg\max_{x\in S} \overline{U}_i(x) && (2.8)\\ &= \arg\max_{x\in S} x \cdot \overline{w}_i && (2.9)\end{aligned}$$

Claim The first element chosen by the greedy heuristic is identical to the best-1 selection for $l = 1$ (and thus equivalent to the first element chosen by the naive algorithm)
Proof For the purpose of this proof, assume the following submodular utility function:

$$C(T, i) = \begin{cases} 0 & \text{if } T = \emptyset, \\ \max_{x\in T}\ U_i(x) & \text{otherwise} \end{cases} \qquad (2.10)$$

In the first iteration of the submodular greedy algorithm, the solution set is equal to $\emptyset$ ($T_1 = \emptyset$) The solution chosen is thus:

$$\begin{aligned}\arg\max_{x\in S} \sum_i p_i \max(C(T_1, i), U_i(x)) &= \arg\max_{x\in S} \sum_i p_i \max(C(\emptyset, i), U_i(x)\\ &= \arg\max_{x\in S} \sum_i p_i \max(0, U_i(x))\\ &= \arg\max_{x\in S} \sum_i p_i U_i(x)\\ &= \arg\max_{x\in S} \overline{U}_i(x)\\ &= \arg\max_{x\in S} x \cdot \overline{w}_i\end{aligned}$$

Figure 2.3: Proof of equivalence between first choices of naive and greedy algorithms

```
MINIMUMCOST(RouteSet S, PreferenceVector p)
if $|S| = 0$
  then
      return $\infty$;

minCost $\leftarrow -1$;
for each route $r \in S$
do
    if $getCost(p, r) < minCost \;||\; minCost = -1$
      then $minCost \leftarrow getCost(p, r)$;


return $minCost$;

```

```
SCORESET(RouteSet S, Population P)
$totalCost \leftarrow 0$;
for each preference vector $p \in P$
do
    $totalCost \leftarrow totalCost +$ MINIMUMCOST$(S, p)$;

return $totalCost$;

```

Figure 2.4: Helper functions for Greedy and Hill climbing algorithms

```
GreedyBestNRoutes(RouteList R, Population P, int n)
$subModularSet \leftarrow \emptyset$
while $|subModularSet| < n$
do
    $minRouteCost \leftarrow -1;$
    $minRoute \leftarrow NIL;$
    for each route $r \in R$
    do
        $routeCost \leftarrow 0;$
        for each preference vector $p \in P$
        do
            $minCost \leftarrow$ minimumCost$(subModularSet, p);$
            $routeCost \leftarrow routeCost + MIN(minCost, getCost(p, r));$

        if $routeCost < minRouteCost \,||\, minRouteCost = -1$
          then $minRouteCost \leftarrow routeCost;$
                $minRoute \leftarrow r;$


    $subModularSet \leftarrow subModularSet \bigcup minRoute;$

return $subModularSet;$
```

Figure 2.5: Greedy Selection Pseudocode

2.10 Implementation of the Hill-Climbing Algorithm

Our implementation of hill-climbing works as follows: An initial result set is chosen at random from our set of candidate solutions. The least beneficial route in this set is determined by computing average marginal costs. The element with the least average marginal cost is replaced by the first improvement found in the set of candidate solutions. We iterate this process until no further improvements are found (within some time limit). This procedure is augmented with random restarts. After some maximum number of iterations, the algorithm terminates and returns the best solution found.

```
HillClimbingBestNRoutes(RouteList R, Population P, int n, int MaxIterations,int patienceFactor)
$totalIterations \leftarrow 0$;
$iterationCount \leftarrow 0$;
$currentSet \leftarrow$n routes chosen randomly from R
$bestSet \leftarrow currentSet$
repeat
        $initialCost \leftarrow \textsc{scoreSet}(currentSet, P)$;
        $minimumDifference \leftarrow -1$;
        $worstRoute \leftarrow NIL$;
        for each route $r \in R$
        do
            $subset \leftarrow currentSet \setminus \{r\}$;
            // subModularity guarantees this difference to be $\geq 0$
            $diff \leftarrow \textsc{scoreSet}(subset, P) - initialCost$;
            if $diff < minimumDifference \;||\; minimumDifference = -1$
              then $minimumDifference \leftarrow diff$;
                   $worstRoute \leftarrow r$;


        if $Random(0,1) < e^{\frac{-patienceFactor}{iterationCount+1}}$
          then $currentSet \leftarrow$n routes chosen randomly from R
                $iterationCount \leftarrow 0$;

          else  for each route $r \in R$ in order, starting at a random point
                do
                    $newset \leftarrow currentSet \bigcup \{r\} \setminus \{worstRoute\}$;
                    if $\textsc{scoreSet}(newset, P) < initialCost$
                      then
                            $improvementRoute \leftarrow r$;
                            $break$;

                $currentSet \leftarrow currentSet \bigcup \{improvementRoute\} \setminus \{worstRoute\}$;

        if $\textsc{scoreSet}(currentSet, P) < \textsc{scoreSet}(bestSet, P)$
          then
                $bestSet \leftarrow currentSet$;
                $iterationCount \leftarrow 0$;

        $iterationCount \leftarrow iterationCount + 1$;
        $totalIterations \leftarrow totalIterations + 1$;

  until $totalIterations > MaxIterations$
return $bestSet$;

```

Figure 2.6: Hill Climbing Pseudocode

Chapter 3

Results

3.1 Preliminary Results

Data for 10, 15, and 20-mile tasks are included in Tables 3.1, 3.2, and 3.3, respectively. Data for the "planar" populations are omitted, as results for these populations are very similar, all showing strong dominance of the greedy and hill-climbing algorithms.

	Algorithm		
Population	Greedy	HillClimbing	Naive
basis	$\mu = 5.38$ $\sigma = 23.3$	$\mu = 5.39$ $\sigma = 23.3$	$\mu = 9.82$ $\sigma = 49.1$
uniform independent	$\mu = 24.57$ $\sigma = 121.3$	$\mu = 25.05$ $\sigma = 122.8$	$\mu = 24.79$ $\sigma = 123$
uniform correlated	$\mu = 24.15$ $\sigma = 116.7$	$\mu = 24.25$ $\sigma = 116.8$	$\mu = 25.24$ $\sigma = 125.4$
normal independent	$\mu = 25.02$ $\sigma = 124.1$	$\mu = 25.02$ $\sigma = 124.1$	$\mu = 25.03$ $\sigma = 124.2$
normal correlated	$\mu = 24.8$ $\sigma = 122.5$	$\mu = 24.82$ $\sigma = 122.5$	$\mu = 25.1$ $\sigma = 124.9$

Table 3.1: Mean & standard deviation of cost, 552 10-mile tasks.

Our data show that the algorithms which optimize for diversity outperform the naive algorithm when there is diversity within the population. This is made very clear by the mean performance of the greedy algorithm relative to the naive for the basis population.

The greedy and hill-climbing algorithms express their superiority over the naive approach when the population has some underlying segmented structure of which the algorithms can take advantage. This is made very clear by the scores obtained with the basis population.

Uniform random populations are essentially populations with no structure, i.e., no distinguishable population segmentation. Thus, it is understandable that optimizing for the typical case, the approach taken by the naive algorithm, performs almost as well as the greedy and hill-climbing

	Algorithm		
Population	Greedy	HillClimbing	Naive
basis	$\mu = 4.53$ $\sigma = 22$	$\mu = 4.54$ $\sigma = 22$	$\mu = 9.72$ $\sigma = 59.5$
uniform independent	$\mu = 23.24$ $\sigma = 138.8$	$\mu = 23.25$ $\sigma = 138.8$	$\mu = 24.13$ $\sigma = 147.1$
uniform correlated	$\mu = 23$ $\sigma = 136.4$	$\mu = 23.15$ $\sigma = 136.8$	$\mu = 24.58$ $\sigma = 150.1$
normal independent	$\mu = 24.21$ $\sigma = 147.4$	$\mu = 24.22$ $\sigma = 147.4$	$\mu = 24.33$ $\sigma = 148.4$
normal correlated	$\mu = 23.42$ $\sigma = 139.9$	$\mu = 23.44$ $\sigma = 140.1$	$\mu = 24.45$ $\sigma = 149.4$

Table 3.2: Mean & standard deviation of cost, 512 15-mile tasks.

	Algorithm		
Population	Greedy	HillClimbing	Naive
basis	$\mu = 2.78$ $\sigma = 12.5$	$\mu = 2.79$ $\sigma = 12.5$	$\mu = 4.02$ $\sigma = 18.7$
uniform independent	$\mu = 9.98$ $\sigma = 45.9$	$\mu = 10$ $\sigma = 45.9$	$\mu = 10.03$ $\sigma = 46.1$
uniform correlated	$\mu = 9.91$ $\sigma = 45.9$	$\mu = 10$ $\sigma = 45.9$	$\mu = 10.19$ $\sigma = 47$
normal independent	$\mu = 10.09$ $\sigma = 46.5$	$\mu = 10.1$ $\sigma = 46.5$	$\mu = 10.1$ $\sigma = 46.5$
normal correlated	$\mu = 10.07$ $\sigma = 46.4$	$\mu = 10.08$ $\sigma = 46.4$	$\mu = 10.13$ $\sigma = 46.7$

Table 3.3: Mean & standard deviation of cost, 526 20-mile tasks.

methods. It is important to mention that, while mean scores for the randomized populations are very close, we did not observe a single trial where the naive algorithm outperformed the greedy algorithm.

Profiles with costs normally distributed form a population that contains "structure," due to the concentration of probability density about the mean. These populations correspond precisely to the case in which the naive approach is reasonable. This observation is supported by our data: the differences between scores of the greedy and hill-climbing techniques and the naive approach for the normal populations are smaller than for their uniform counterparts.

From this argument it ought to follow that adding correlation between cost elements would improve the relative performance of our algorithms by introducing, in a rough approximation, a segmentation of the population determined by the values of these correlated costs. This observation, too, is borne out by our data, where the performance gaps between our methods and the naive approach for the correlated populations are larger than for corresponding independent populations.

While we did not observe the hill-climbing algorithm outperforming the greedy algorithm in our experiments, there are certainly situations where this outcome could arise: Consider, for example, choosing a subset of size 2 from a set of 3 options with attribute vectors $r_1 = (0, 1), r_2 = (1, 0)$, and $r3 = (0.4, 0.4)$ to satisfy users with cost vectors $A = (0, 1), B = (1, 0)$. In its first iteration, the greedy algorithm makes the following association betwen route alternatives and cost: $r_1 = 1, r_2 = 1, r_3 = 0.8$, and chooses r_3. In the second iteration, it scores the remaining alternatives $r_1 = 0.4, r_2 = 0.4$. Suppose we break ties arbitrarily, and the result set is $\{r_2, r_3\}$. This set has a total cost of 1.4, while the optimal result set is clearly $\{r_1, r_2\}$ with cost 0.

3.1.1 Technical Details

Our experiments were coded in Java. We tested the performance of the algorithms for each of our 15 test populations, over roughly 500 trials of each task length. Our "basis population" was of size 5 and our "planar populations" were of size 2. (In these special cases, larger population sizes with identical proportions would not impact the results). For all other populations, we drew 1000 samples.

3.2 Further Results

In order to obtain a better understanding of the relative performance of the Naive, Greedy, and Hill-Climbing algorithms, we conducted further experiments, holding a set of tasks constant, and testing each algorithm against 500 populations of each distribution type. Each population, again, was 1000 samples from the particular population distribution.

The difficulty of satisfying a user population may vary from task to task; this approach prevents such variance from affecting our analysis of relative algorithm performance.

In addition, our implementation of hill-climbing described above was a first approximation to an algorithm we believe should, in certain cases, outperform the greedy heuristic. In an attempt to

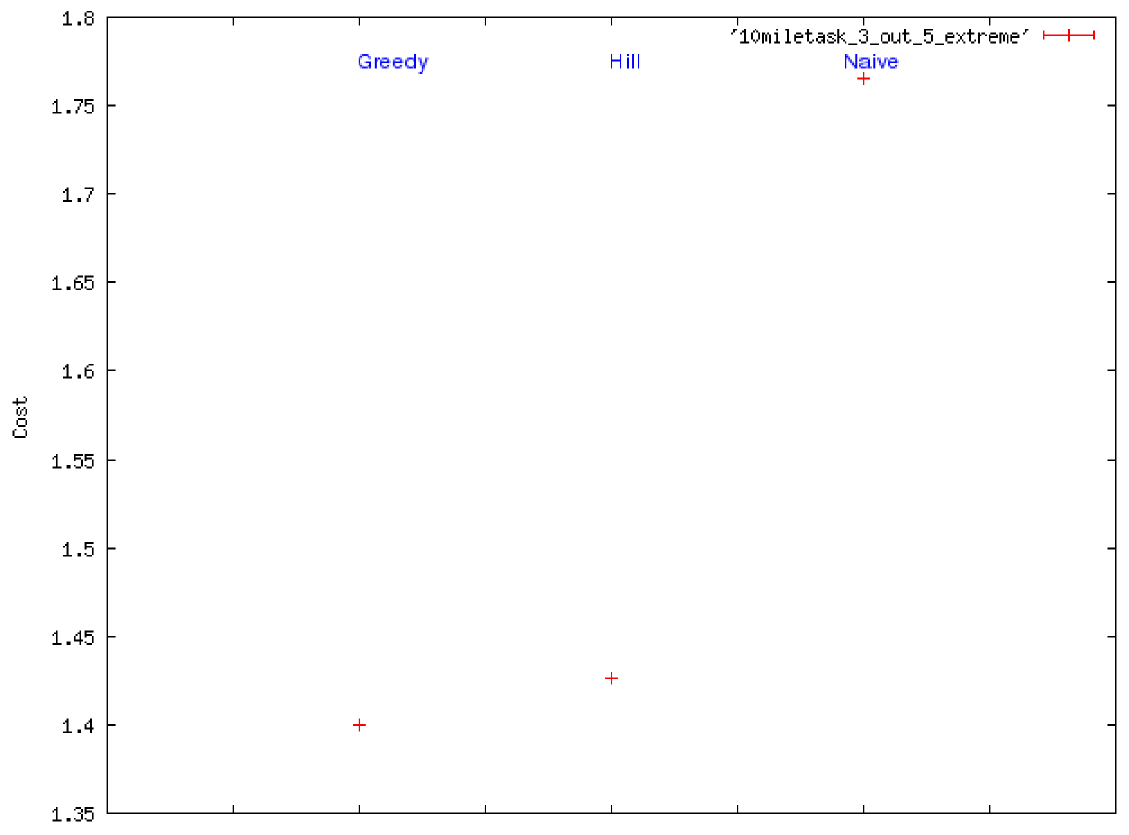

Figure 3.1: Algorithm performance with basis population on a 10-mile task

revise the hill-climbing algorithm, we modified the neighborhood exploration strategy to investigate truly random moves. In our revised hill-climbing, we do not necessarily eliminate the route with least marginal cost, but eliminate a random route. We do not always replace the eliminated route with a strictly improving route, but rather with a random route. We also allow, with some probability which decays as the search progresses, swaps which do not improve our objective value. This technique is commonly known as simulated annealing.

This second set of experiments showed that, while our proposed methods always significantly outperformed the naive approach with the basis population, their relative performance on randomized populations was highly dependent upon the particular task at hand.

An example of typical algorithm performance on the basis population is included in Figure 3.1. The relative algorithm performance against the basis population demonstrated by this figure is consistent across all tasks, and all lengths of task, which we tested.

This particular task shows one of three interesting patterns of algorithm performance observed with randomized populations. The pattern shown by this task is such that no real advantage is demonstrated by the heuristic or hill climbing methods over the naive approach (Figures 3.2, 3.3, 3.4, 3.5) We believe that such situations occur when there is insufficient variety within the space of

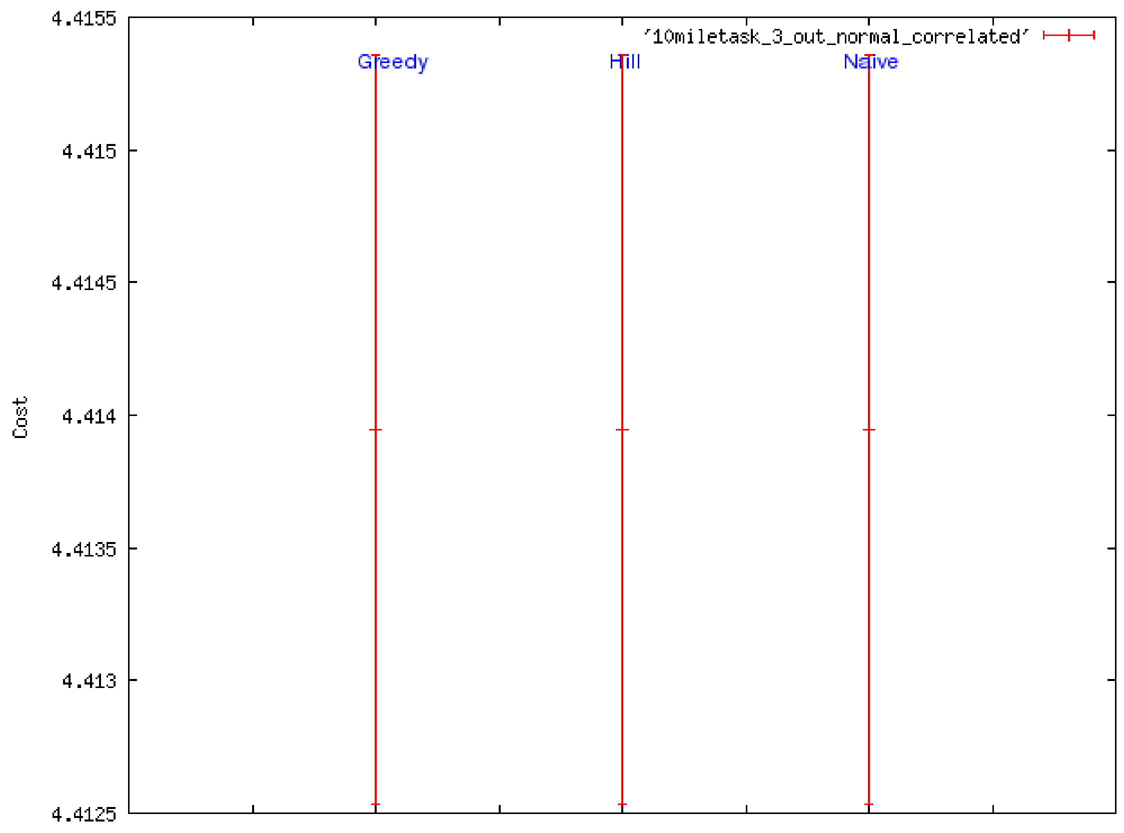

Figure 3.2: 95% confidence intervals over algorithm performance with normally distributed, correlated population on a 10-mile task

routing task solutions. This could easily happen if all paths between endpoints of a given task have very little choice, or possibly no choice, of alternative roads to use. When the set of solutions from which to choose shows excessive self-similarity, and users express varied preferences, there is little that can be done to satisfy them. Note that across figures 3.2, 3.3, 3.4, and 3.5 the confidence intervals span less than 0.01 units of cost. These confidence intervals represent 500 samples from each population distribution, indicating that for this task, with randomized preferences, all of the algorithms are essentially returning the same subset, with the same cost to population.

One might then ask: if the set of solutions was so self-similar that it prevented any one algorithm from outperforming another, why the large gap in scores for the basis population? The basis population differs from any of the randomized populations in that it is such an extreme example, with segments of the population expressing strong, mutually disjoint preferences. Even though many routes in the set may be similar, the extremes always exist. For tasks such as these, with self-similar solution sets, the performance gap on the basis population is not so much due to the greedy heuristic being able to select from a wide variety of routes, but instead due to the naive approach losing population segmentation information in its averaging.

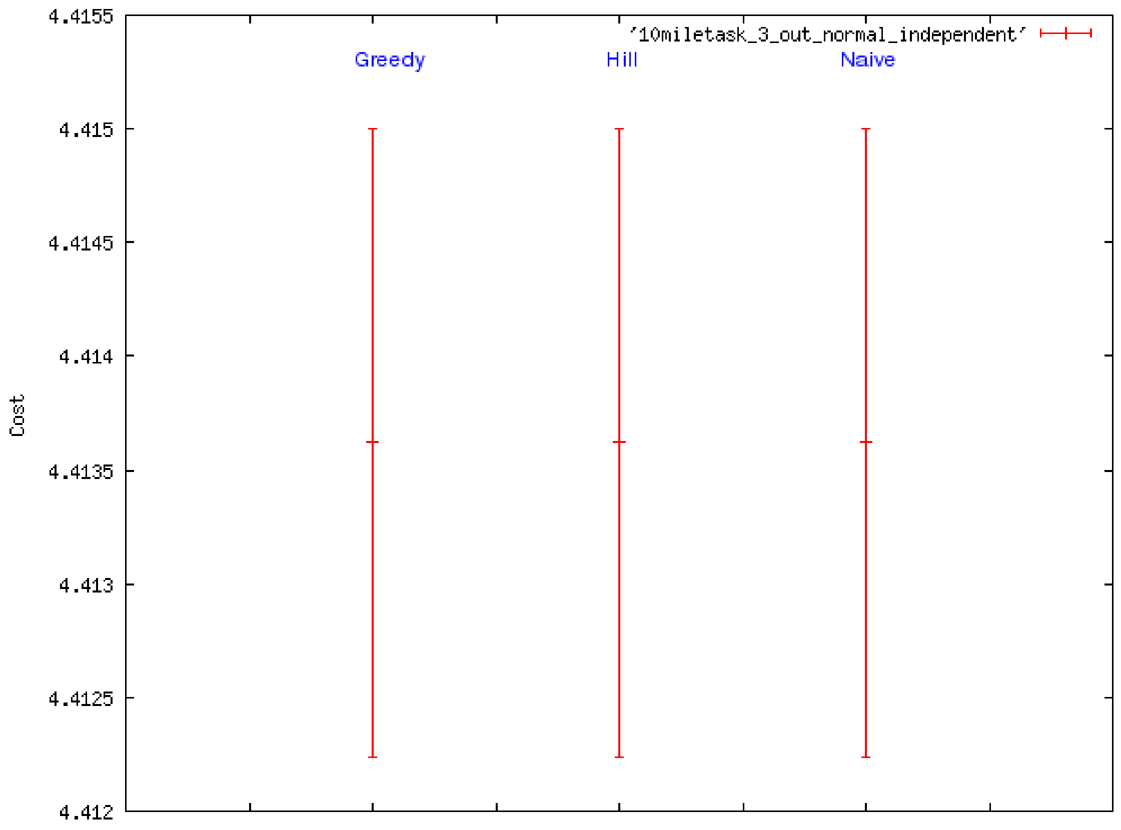

Figure 3.3: 95% confidence intervals over algorithm performance with normally distributed, independent population on a 10-mile task

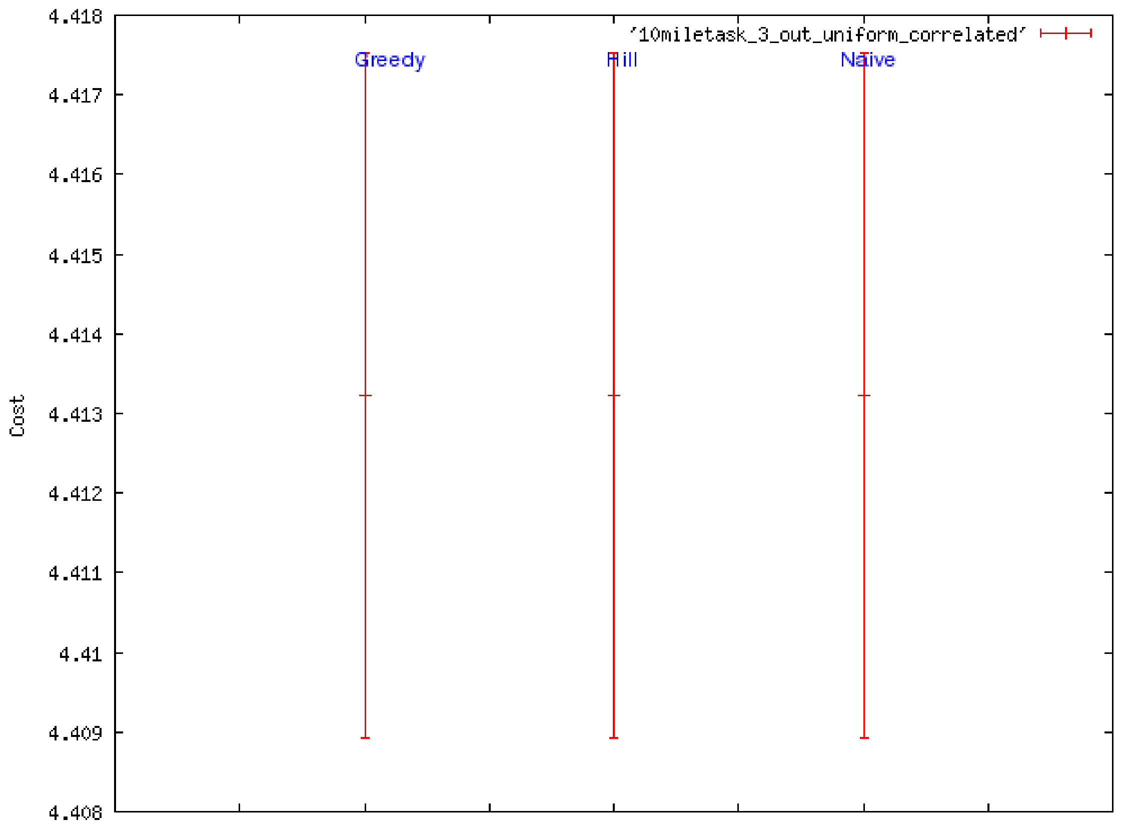

Figure 3.4: 95% confidence intervals over algorithm performance with uniformly distributed, correlated population on a 10-mile task

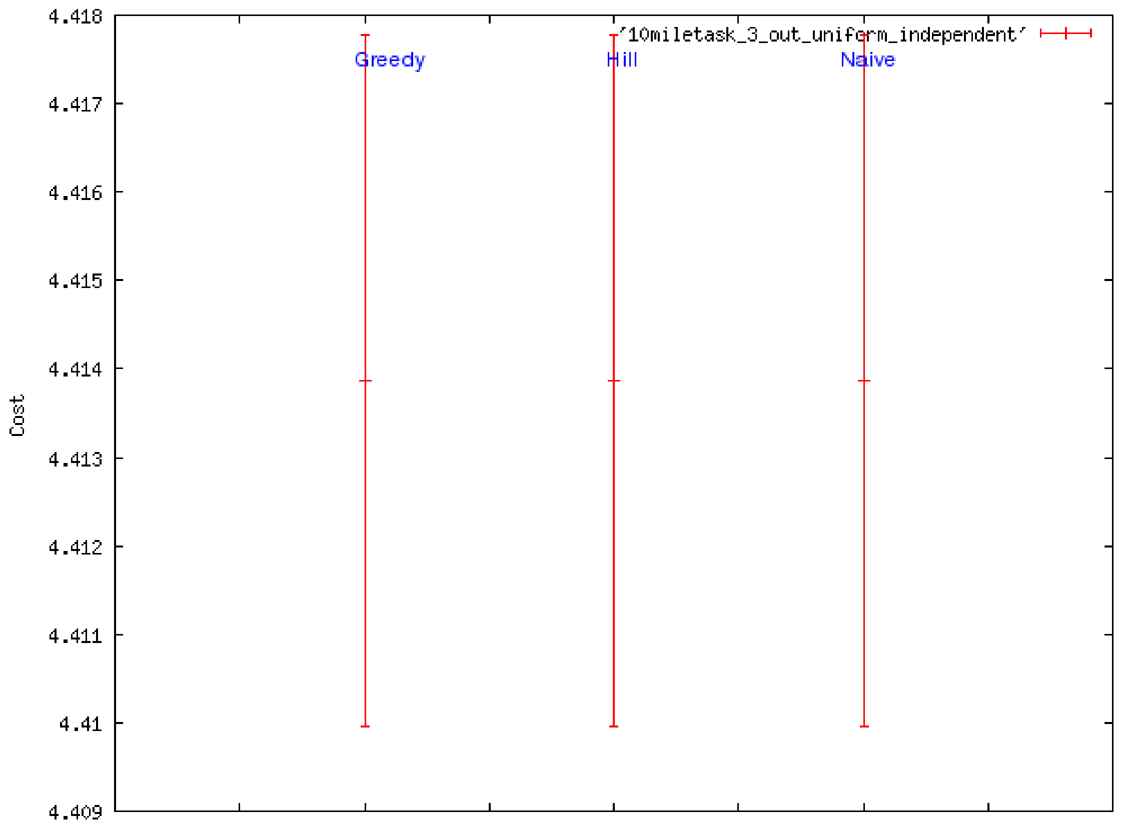

Figure 3.5: 95% confidence intervals over algorithm performance with uniformly distributed, independent population on a 10-mile task

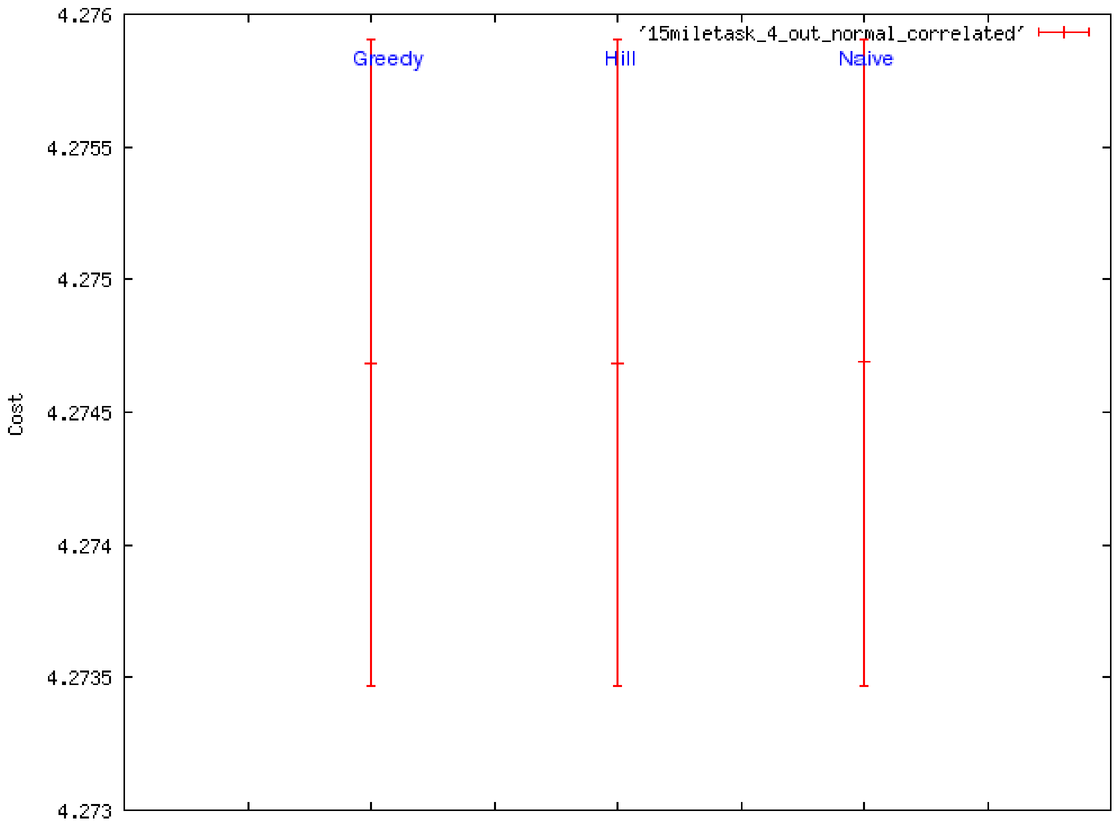

Figure 3.6: 95% confidence intervals over algorithm performance with normally distributed, correlated population on a 15-mile task

The second interesting pattern observed in our data was a pattern of tasks where no clear benefit was given by the greedy and hill-climbing algorithms over the naive approach when considering normally distributed populations, yet a benefit was given when considering uniformly distributed populations.

The naive algorithm's acceptable performance with normally distributed population preferences is not surprising, as explained above in the preliminary results. Normally distributed populations will tend to cluster strongly about the mean, thus exhibiting fewer distinct segments and diluting the effect of diversity in reducing cost to the population. With uniformly distributed population preferences, however, it is more likely that there will be segments within the population with sufficiently distinct preferences, resulting in a greater probability that some of the variety within the solution set can be matched to the variety within population preferences.

Algorithm performance for one such 15-mile task is shown in figures 3.6, 3.7, 3.8, and 3.9.

The third and final pattern observed in our data is that in which the greedy and hill-climbing algorithms show a definite advantage over the naive algorithm in all cases. In contrast to the second pattern described above, where normally distributed populations (i.e. a population of very similar

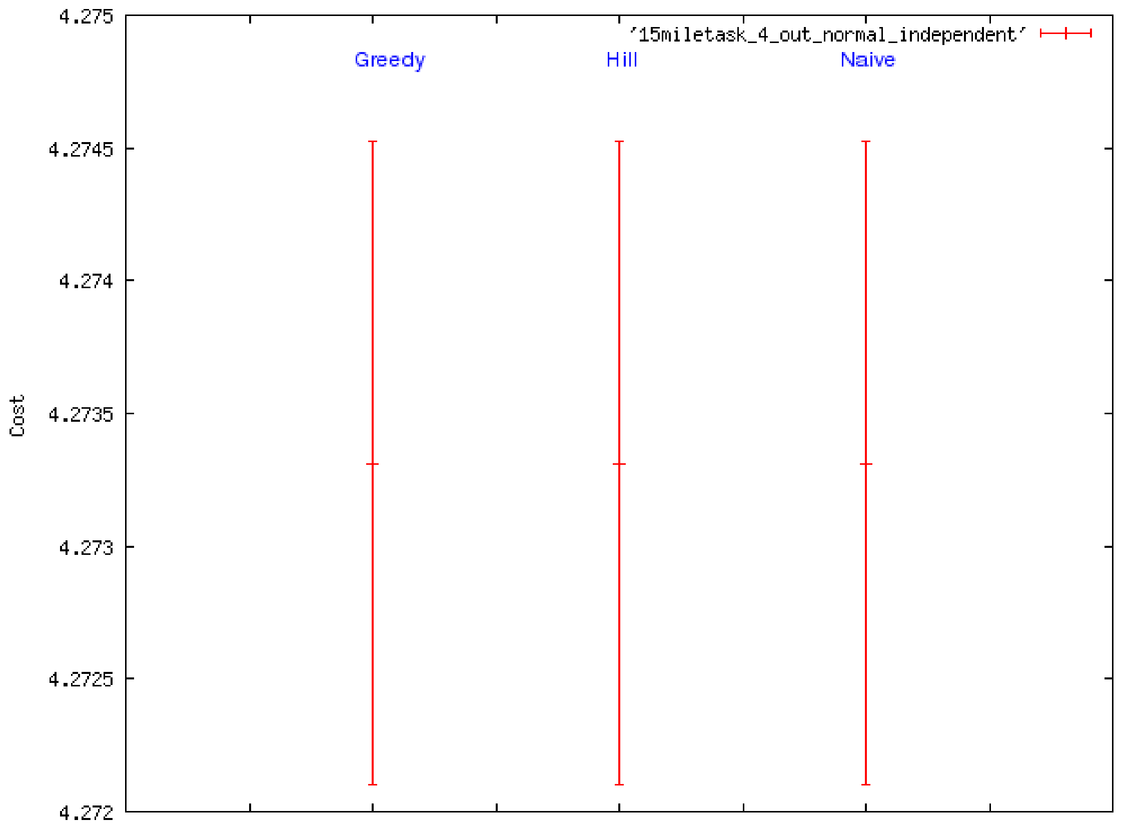

Figure 3.7: 95% confidence intervals over algorithm performance with normally distributed, independent population on a 15-mile task

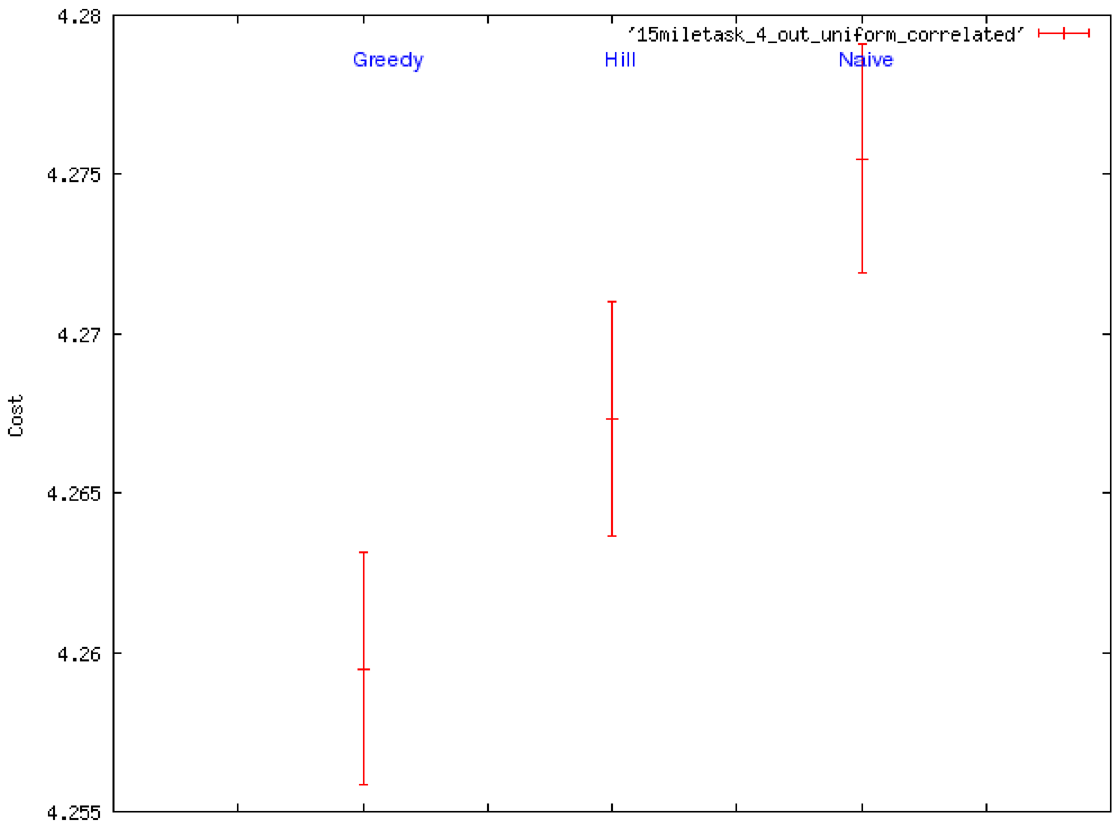

Figure 3.8: 95% confidence intervals over algorithm performance with uniformly distributed, correlated population on a 15-mile task

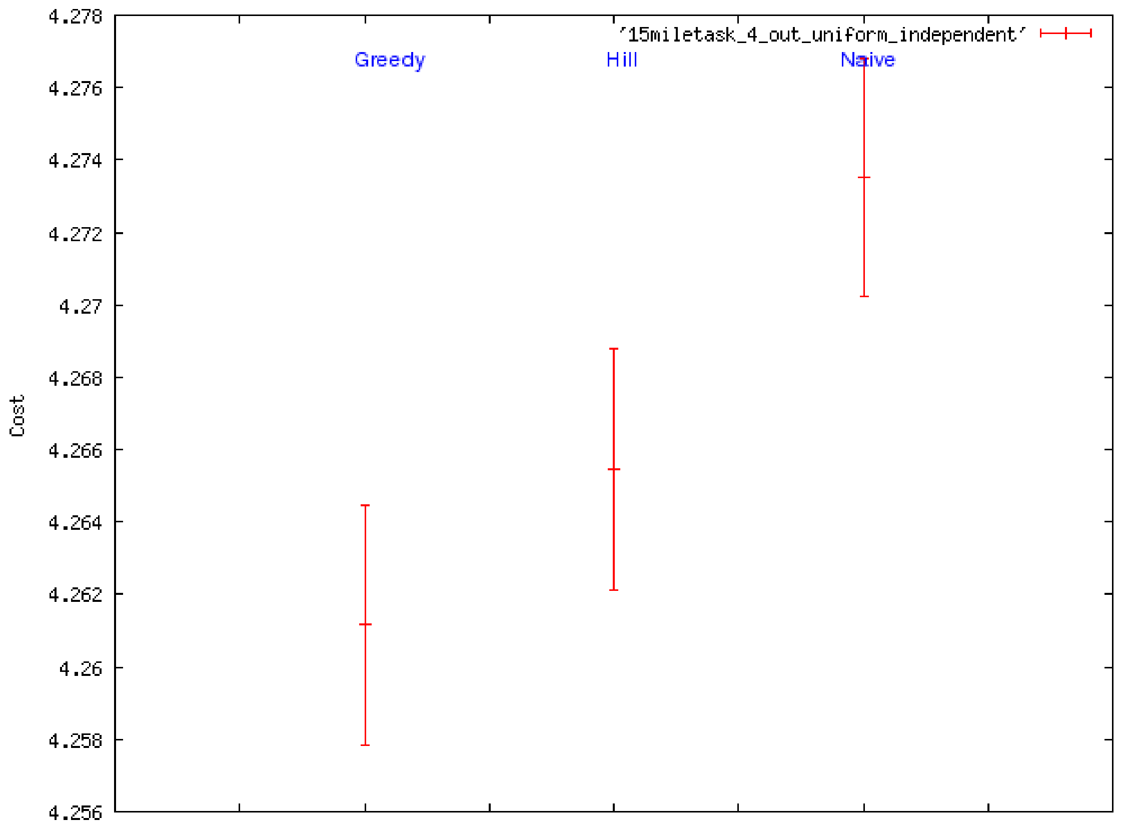

Figure 3.9: 95% confidence intervals over algorithm performance with uniformly distributed, independent population on a 15-mile task

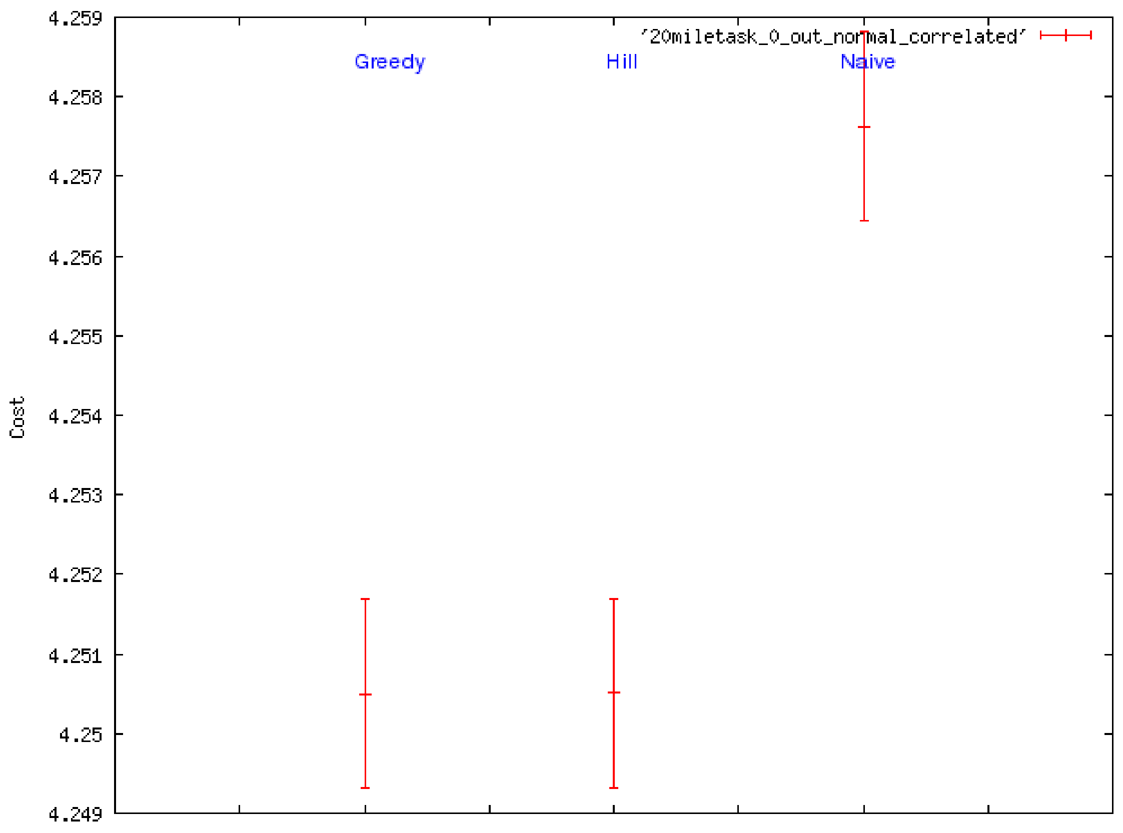

Figure 3.10: 95% confidence intervals over algorithm performance with normally distributed, correlated population on a 20-mile task

preferences) are no better satisfied by the greedy and hill-climbing algorithms than by the naive, solution sets for tasks fitting this pattern express so much variety that routes satisfying even highly self-similar population segments can be chosen, providing a solution closely matching population preferences.

Intuitively, the difference is as follows: tasks matching the second pattern show sufficient variety to match larger scale separations among population segments, such as those in the uniformly distributed populations, but insufficient variety to match highly similar segments, such as those that would be seen in normally distributed populations. Tasks matching the third pattern show enough variety among solutions that even the highly similar segments in a normally distributed population can be satisfied.

Algorithm performances on one such 20-mile task are shown in figures 3.10, 3.11, 3.12, and 3.13.

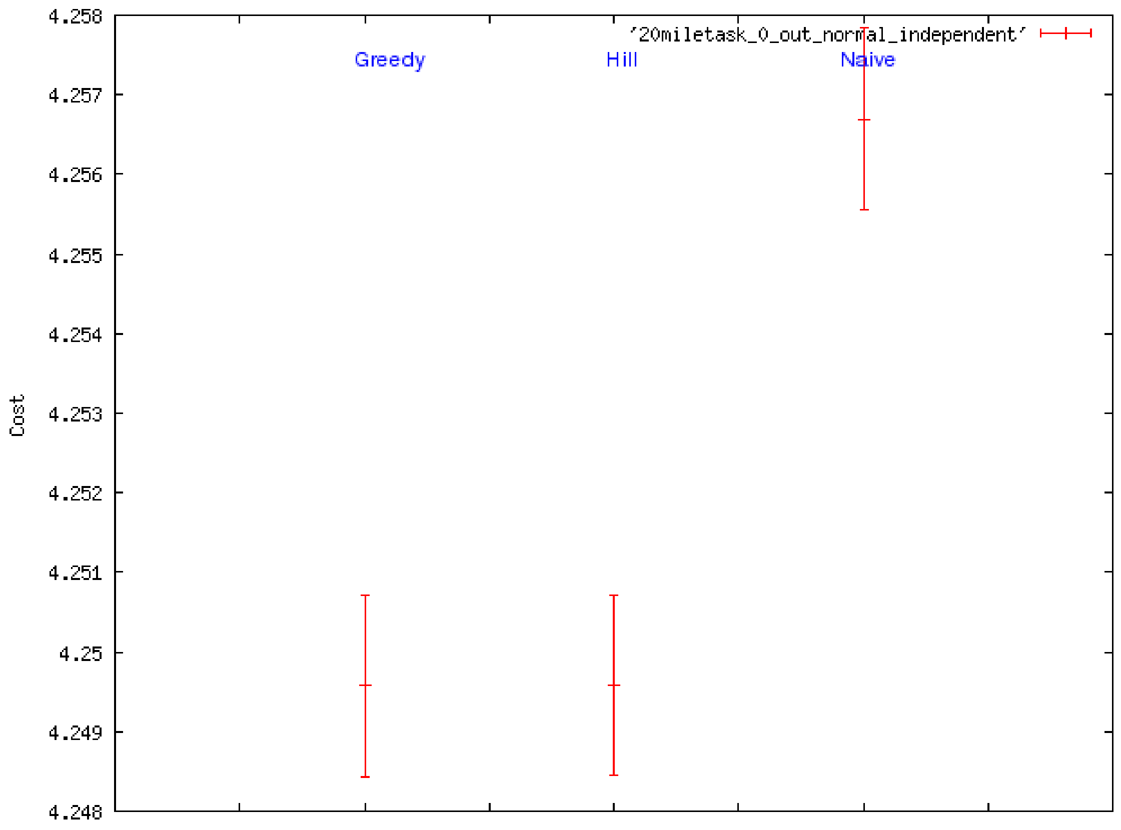

Figure 3.11: 95% confidence intervals over algorithm performance with normally distributed, independent population on a 20-mile task

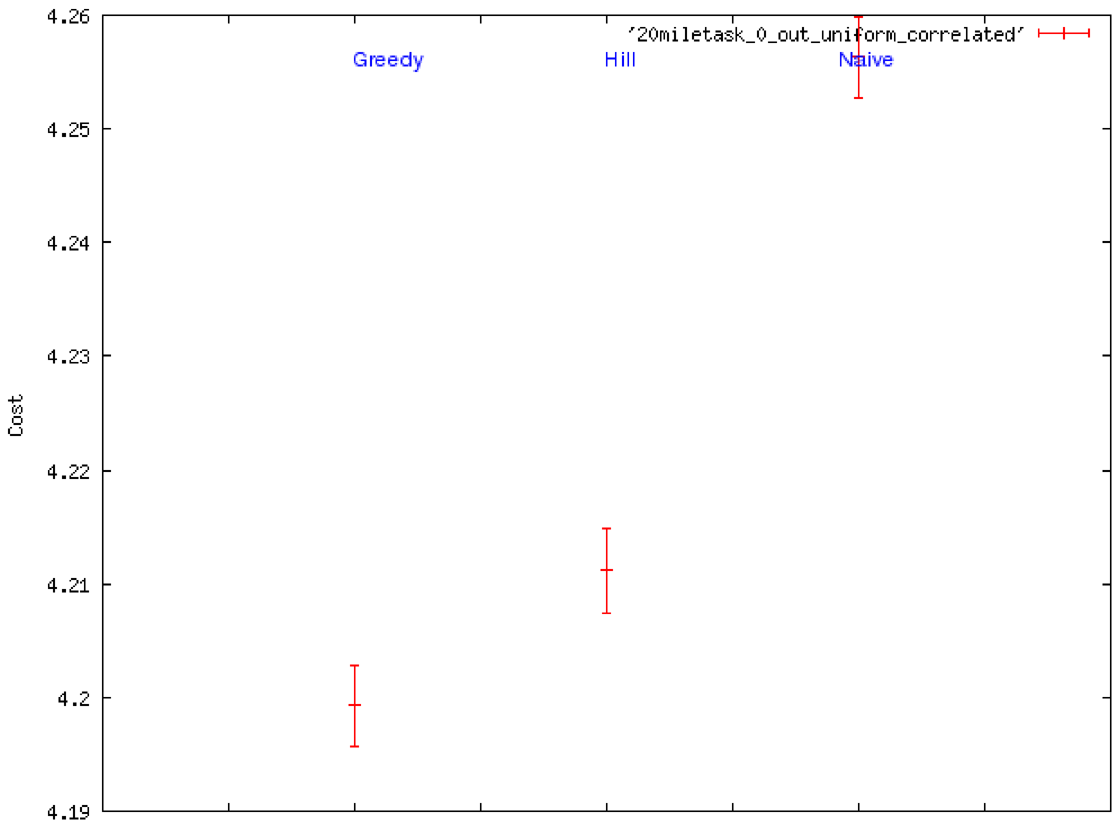

Figure 3.12: 95% confidence intervals over algorithm performance with uniformly distributed, correlated population on a 20-mile task

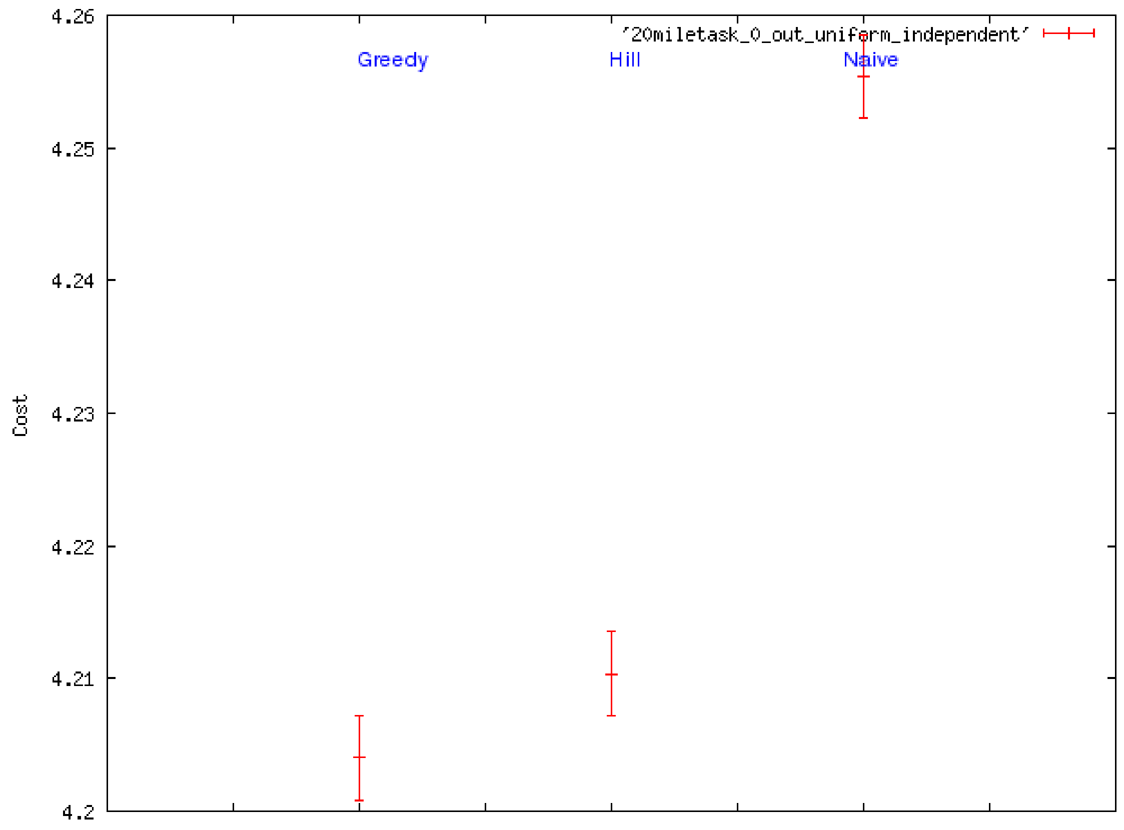

Figure 3.13: 95% confidence intervals over algorithm performance with uniformly distributed, independent population on a 20-mile task

Chapter 4

Related Work

An in-depth investigation of the space of driving directions solutions was carried out in [7]. Previous work on context-dependent driving directions, such as [5, 12, 8] concern adaptive learning and case-based reasoning techniques for providing better routing solutions to a single user. Recommendation systems proposed by [9] require intervention after result presentation to build an understanding of the particular user performing the search. Systems such as those proposed in [3, 4] rely on similarity metrics to eliminate redundancy and introduce diversity in result sets, but set forth explicit goals of deviating little from an original user query. *The most significant difference between our approach to the meta-choice problem and others is that diversity in the choice set, when it arises, is not a result of optimizing with respect to an ad-hoc combination of similarity and diversity metrics, but rather it is a direct function (i.e., reflection) of the diversity in user preferences.*

Chapter 5

Future Work

One of our goals is to build a transportation system (like `orbitz.com` but unlike `maps.yahoo.com`) that provides users with a variety of travel options, rather than a single option. Ideally, in response to a query about how to get from Providence, RI to Cambridge, MA, the system would respond with driving directions, bus routes, train routes, etc. In order to do so, there are a number of research directions yet to be explored.

Hill-Climbing

Our revised hill-climbing algorithm is an improvement over our first hill-climbing formulation, but still fails to outperform the greedy heuristic algorithm. We believe that a weakness of the hill-climbing algorithm is its use of what is essentially the greedy heuristic objective in its scoring function. This causes the hill climbing algorithm to perform in a manner very similar to the greedy heuristic and then stagnate at what is essentially a plateau. A different objective function could very well lead to hill-climbing outperforming the greedy heuristic, perhaps replacing elements in proportion to their individual scores.

Clustering

One further technique we plan to explore is clustering. To solve the selection problem, clustering can be performed on inputs, on outputs, or on both. Given an input of m contexts, we can cluster to arrive at l representative contexts and then solve the context-dependent selection problem l times with respect to these representatives. Alternatively, we can cluster the outputs: solve the deterministic selection problem that arises in each context, and then cluster solutions to arrive at a selection of size l. The complexity of the latter procedure exceeds that of the former as it requires $m > l$ optimization problems to be solved. Intermediate solutions are also possible, e.g., one might first cluster the input data into $l' > l$ clusters, then optimize deterministically for each representative, and finally cluster the set of l' outputs into a selection of size l.

Efficiency

Although we reported that the heuristic methods outperformed the naive approach in terms of raw scores in our experiments, we did not report runtimes. In fact, the runtimes for the heuristic methods far exceeded those of the naive approach. One bottleneck for these methods is the generation of the candidate solution set. We used multi-A^*search with an inadmissible heuristic to expedite this process, but depending on the routing task, our implementation exhibited great variability in runtimes.

With certain IR systems, the candidate solution set could prove prohibitively large for direct application of our techniques. We plan to investigate how techniques such as those described in [1] could be employed to reduce to a representative sample the number of candidates considered.

Preference Representation

While our experimental setup was sufficient to demonstrate the superiority of our heuristic methods over the naive approach to the selection problem, there are (at least) two shortcomings that must be overcome before a navigation system could be built based on our contextual preference representation. First, our underlying model of context-dependent preferences was based on a weighted additive utility function, a representation that is well-known to be inadequate for representing human preferences. Second, our choice of population distributions was arbitrary. We experimented with 15 different populations, and we established the superiority of the heuristic methods over the naive approach in all 15 cases; but it remains to conduct user studies to determine realistic context distributions in various domains (e.g., travel or shopping), either for an individual user or for a population of users.

User representation

An important metric for our suggested methods will be their performance given real-world distributions obtained from user studies. We believe that real-world user populations will demonstrate a segmented underlying structure better handled by our proposed methods than by the naive approach. Further, we would like to investigate more complex cost functions, as linear combinations could very well be inadequate to characterize a user's desires. For example, it is logical to assume some users assign no cost to a certain attribute up to a certain threshold, after which point their annoyance might grow exponentially.

We conjecture that these populations with significant correlation between costs attributed to distance and time match a segments of the population of real world users, where those users who assess a high cost to the length of a route would also be relatively averse to routes that take a longer amount of time. It is entirely possible, however, that there are segments for which these costs are negatively correlated. Take as one example a user on vacation with a mileage limit on their rental car; this user might assess a high cost to distance and yet not mind routes that are not time-efficient. On the other end of the spectrum, an ambulance driver might be willing to take a longer route (say, on a highway) provided the higher speed limit along the route would reduce the time to destination.

Chapter 6

Conclusion

Since the user is operating in one of many possible contexts, the IR system's objective is to present a set of options, which are useful in various contexts, particularly those contexts which are most likely to be realized. An optimal solution to meta-choice/selection should contain a *diverse* set of options, with the various options presented reflective of the user's various utility functions, which can arise in the various contexts.

Ultimately, solutions to the meta-choice problem should increase social welfare by enabling people to find whatever they are seeking, be it information, product, route, flight, graduate student, advisor, or spouse.

Bibliography

[1] A. Anagnostopoulos, A.Z. Broder, and D. Carmel. Sampling search-engine results. In *Proceedings of the 14th International World Wide Web Conference*, 2005.

[2] John Birge and Francois Louveaux. *Introduction to Stochastic Programming.* Springer, New York, 1997.

[3] K. Bradley and B. Smyth. Improving recommendation diversity. In *Proceedings of the Twelfth National Conference in Artificial Intelligence and Cognitive Science*, pages 75–84, 2001.

[4] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of SIGIR-98*, 1998.

[5] K. Haigh and M. Veloso. Route planning by analogy. In *Proceedings of the International Conference on Case-Based Reasoning*, pages 169–180, 1995.

[6] R. L. Keeney and H Raiffa. *Decisions with Multiple Objectives.* John Wiley and Sons, New York, 1976.

[7] C.M. Leroy. Exploration of the routing task domain. Master's thesis, 2003.

[8] L. McGinty and B. Smyth. Collaborative case-based reasoning:applications in personalised route planning. In *Proceedings of the International Conference on Case-Based Reasoning*, pages 362–376, 2001.

[9] L. McGinty and B. Smyth. Comparison-based recommendation. In *Proceedings of the Sixth European Conference on Case-Based Reasoning*, pages 575–589, 2002.

[10] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14:265–294, 1978.

[11] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions-ii. *Mathematical Programming Studies*, 8:73–87, 1978.

[12] C. Rogers, S. Fiechter and P. Langley. An adaptive interactive agent for route advice. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[13] A. Tversky and I. Simonson. Context-dependent preferences. *Management Science*, 39:1179–1189, 1993.

[14] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, Princeton, 1944.