DISCRETE OPTIMIZATION

Megan Bryant
mrbryant@email.wm.edu

Department of Mathematics, The College of William and Mary, P.O. Box 8795, Williamsburg, VA 23187

Executive Summary

The files you will need to implement the simulated annealing code from the Numerical Recipes text may be downloaded from Dr. Rex Kincaid's homepage: http://www.math.wm.edu/rrkinc.

A copy of this section of Numerical Recipes is available at: http://www.nrbook.com/a/bookcpdf.php.

A copy of 'Optimization by Simulated Annealing: An Experimental Evaluation; Part 1, Graph Partitioning' by Johnson et. al. is available at: http://vis.lbl.gov/ aragon/pubs/annealing-pt1.pdf and http://vis.lbl.gov/ aragon/pubs/annealing-pt2.pdf.

1

a.) *Explain what the parameters T, NOVER, NLIMIT, and TFACTOR mean in the Numerical Methods algorithm.*

The following table includes the above definitions as well as definitions used in part **b.)**.

Term	Definition
T	The temperature.
NOVER	The maximum number of paths (moves) examined at any T.
NLIMIT	The maximum number of successful path changes allowed at any T.
TFACTOR	The cooling ratio r for temperature.
INITPROB	Used to determine the what fraction of moves is accepted.
TEMPFACTOR	The descriptive name for the cooling ratio r.
SIZEFACTOR	A fixed value for instance sizes.
MINPERCENT	Used to determine whether the annealing run is frozen.
STARTTEMP	The starting temperature.
N	The expected neighborhood size.
L	The temperature length, set to $N * SIZEFACTOR$.
ALPHA	Used to remove unneeded trials from beginning of schedule.

b.) *How do these parameters relate to the parameters SIZEFACTOR*N, TEMPFACTOR, STARTTEMP, and MINPERCENT in the Johnson et. al. algorithm?*

It is evident from the definitions in part **a.)** that NOVER and $L = SIZEFACTOR * N$ serve similar functions in that they both limit the number of moves examined at each temperature. However, our code has set $NOVER = 100 * ncity$, which is not directly equivalent. Therefore, while they serve the same purpose, the approach is different.

The parameters TEMPFACTOR and TFACTOR serve identical purposes as well as they both act as the cooling ratio for the temperature T.

The parameters STARTTEMP and INITPROB are interrelated in that if STARTEMP is set, the trial run for INITPROB (to determine starting temperature) is omitted. INITPROB itself determines what fractions of moves are accepted and can thus be considered a variety of 'cutoff' parameter. The NLIMIT parameter is functioning as a CUTOFF parameter for the Numerical Methods code (it is termed CUTOFF in the SA Johnson et. al. paper). The gains achieved by using a CUTOFF parameter such as NLIMIT can be achieved using only INITPROB = 0.4.

There is no direct equivalent to the parameter MINPERCENT in the Numerical Recipes TSP code. Simulated Annealing uses the concept of MINPERCENT to terminate a search when it becomes frozen, i.e. when fewer than MINPERCENT paths have been accepted. The Numerical Recipes code terminates instead by reaching the maximum number of iterations or there are no successful moves at some time T.

c.) *In the Numerical Methods simulated annealing code, when are accepted moves updated? Describe the move structure that is being employed.*

The Simulated Annealing code that we are examining is being applied to a Traveling Salesman Problem. In this case, there are two move subroutines: transport (TRNSPT) and reverse (REVERSE). It is evident from the structure of the code that an accepted move is updated immediately after being accepted into either of these subroutines.

In order to facilitate our understanding of the SA TSP move structure, it is best to consider the generic starting solution employed by the algorithm.

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$$

We begin the process of selecting a move by marking at random two cities in the solution vector. We will call these marked nodes i and j and, for the purposes of illustration, let $i = 3$ and $j = 6$.

$$1, 2, \underline{3}, 4, 5, \underline{6}, 7, 8, 9, 10$$

Now the SA algorithm must determine which of the move subroutines to execute: TRNSPT or REVERSE. This selection is made randomly and with equal probability. Let us suppose that TRNSPT was selected. The move structure will then generate a new solution by transporting the entire selection from i to j (inclusive) to a new location in the solution vector. This new location is determined by selecting at random with equal probability two nodes not in the selection and inserting the selection between them. In this example, we will insert the transport set between cities 8 and 9.

$$1, 2, 7, 8, 3, 4, 5, 6, 9, 10$$

We have successfully illustrated the transport move. Now, we must examine the other move subroutine: reverse. This is best done by returning to the initial solution with cities 3 and 6 marked.

$$1, 2, \underline{3}, 4, 5, \underline{6}, 7, 8, 9, 10$$

Suppose that the reversal move had been selected. In that case, the subroutine would simply reverse the order of the cities from i to j, inclusive.

$$1, 2, 6, 5, 4, 3, 7, 8, 9, 10$$

This illustrates the entirety of the move structure in our SA TSP code.

2

a.) *Perform four trial runs to the following specifications. Remember: simulated annealing relies on a random number generator that is driven by a SEED value. Therefore, when conducting a run with the same parameter values and the same SEED value, the results will be exactly the same. Thus, to evaluate the performance of a trial, we must compare over a collection of random SEED values.*

Table 1: Test Parameters

Trial	NOVER	NLIMIT	T	TFACTOR
1	100*NCITY	10 *NCITY	0.5	0.90
2	10*NCITY	2*NCITY	0.5	0.90
3	100*NCITY	10 *NCITY	0.1	0.90
4	100*NCITY	10*NCITY	0.5	0.98

For each trial, five replications were run and the results recorded. The seeds that we used are as follows: $31764.0, 35711.13, 17192.32, 01123.58$, and 13213.45. Note that a counter variable was added to facilitate the easy recording of the number of temperatures visited.

Table 2: Trial 1

Run	T Count	Final T	Final Path Length	Optimal Path
1	27	0.0323	28.4733	$(6, 8, 7, 4, 3, 5, 10, 1, 2, 9)$
2	33	0.0172	28.4733	$(3, 5, 10, 1, 2, 9, 6, 8, 7, 4)$
3	32	0.0191	28.4733	$(10, 1, 2, 9, 6, 8, 7, 4, 3, 5)$
4	31	0.0212	28.4733	$(1, 2, 9, 6, 8, 7, 4, 3, 5, 10)$
5	27	0.0323	28.4733	$(6, 9, 2, 1, 10, 5, 3, 4, 7, 8)$
Averages	**30**	**0.0244**	**28.4733**	

Table 3: Trial 2

Run	T Count	Final T	Final Path Length	Optimal Path
1	9	0.2152	28.4733	$(6, 9, 2, 1, 10, 5, 3, 4, 7, 8)$
2	4	0.3645	28.5378	$(10, 5, 3, 4, 6, 8, 7, 9, 2, 1)$
3	6	0.2953	28.5378	$(1, 2, 9, 7, 8, 6, 4, 3, 5, 10)$
4	7	0.2657	28.4733	$(9, 2, 1, 10, 5, 3, 4, 7, 8, 6)$
5	11	0.1743	28.5378	$(4, 3, 5, 10, 1, 2, 9, 7, 8, 6)$
Averages	**7.4**	**0.2630**	**28.5120**	

Table 4: Trial 3

Run	T Count	Final T	Final Path Length	Optimal Path
1	15	0.02288	28.4733	$(3, 5, 10, 1, 2, 9, 6, 8, 7, 4)$
2	8	0.04783	28.4733	$(1, 2, 9, 6, 8, 7, 4, 3, 5, 10)$
3	12	0.03138	28.4733	$(6, 8, 7, 4, 3, 5, 10, 1, 2, 9)$
4	11	0.03487	28.4733	$(10, 1, 2, 9, 6, 8, 7, 4, 3, 5)$
5	15	0.02288	28.4733	$(10, 1, 2, 9, 6, 8, 7, 4, 3, 5)$
Averages	**12.2**	**.03197**	**28.4733**	

Table 5: Trial 4

Run	T Count	Final T	Final Path Length	Optimal Path
1	100	0.06766	28.5379	$(10, 1, 2, 9, 7, 8, 6, 4, 3, 5)$
2	100	0.06766	28.4733	$(4, 3, 5, 10, 1, 2, 9, 6, 8, 7)$
3	100	0.06766	28.4733	$(10, 5, 3, 4, 7, 8, 6, 9, 2, 1)$
4	100	0.06766	28.4733	$(10, 5, 3, 4, 7, 8, 6, 9, 2, 1)$
5	100	0.06766	28.5379	$(10, 1, 2, 9, 7, 8, 6, 4, 3, 5)$
Averages	**100**	**0.06766**	**28.499**	

b.) *Compare and contrast the results of the four trials.*

In order to effectively compare and contrast the results, we must select a base case against which changes are to be evaluated. Since Trial 1 did not involve the changing of any initial parameters other than the seed value for each run, we will consider it the base case for our purposes. Therefore, we can effectively compare the effect of changing individual parameters.

Between Trial 1 and Trial 2, the parameters that change are NOVER and NLIMIT. Trial 2 sees these reduced to $10 * NCITY$ and $2 * NCITY$ respectively. Recall that NOVER is the maximum number of paths examined at any temperature and NLIMIT is the maximum number of successful path changes allowed at any temperature. We see that these parameter changes result in a significant reduction in the number of temperatures examined; the average T count for Trial 1 was 30 versus 7.4 for Trial 2. This decrease in the number of temperature reductions resulted in an increase in variety in the final path lengths observed by Trial 2. Whereas Trial 1 consistently observed path lengths of 28.4733, Trial 2 witness this path length in only 2 of the 5 runs. Overall, the average path length increased in Trial 2 to 28.5120. This leads us to conclude that the reduction in the NOVER and NLIMIT was not preferable as the decrease in observed temperatures results in an increase in variability of observed path lengths. The parameters of Trial 1 can thus be considered more effective.

The temperature parameter is the only parameter that varies between Trial 1 and Trial 3; it is reduced from 0.5 in Trial 1 to 0.1 in Trial 3. We see that this reduction in temperature results in a reduction in the temperature count from an average of 30 in Trial 1 to an average of 12.2 in Trial 2. Remarkably, this reduction is not accompanied by an increase in the final path length; both trials have an average final path length of 28.4733. The reduction in temperature did not have a negative effect on our objective. Assuming that computing resources are a concern, Trial 3 is more beneficial as it significantly reduces the number of temperatures examined. Therefore, we would consider the parameters of Trial 3 to be more efficient.

Finally, the only parameter that differs between Trial 1 and Trial 4 is the TFACTOR, which is increased from 0.90 to 0.98. This results in a dramatically significant increase in the number of temperatures examined from an average of 30 in Trial 1 to an average of 100 in Trial 4. In fact, we note that the number of temperatures counted is the same for each run of Trial 4, implying that it is reaching the cutoff value of 100, not a natural termination. There was also an increase in the average final path length from 28.433 in Trial 1 to 28.499 in Trial 4. This increase in variability is not desirable, especially when considering the increase in computing resources required. Therefore, we would consider the parameters of Trial 1 to be more efficient and effective.

3

a.) *Change the Numerical Methods Simulated Annealing code to reflect what we've learned in the SA Johnson et al. article. Use the concepts of MINPERCENT and INITPROB. This will require the elimination of some parameters (e.g. NLIMIT) and the renaming of some others.*

Altered Code:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "nrutil.h"
#include "rngs.h"
#include "rvgs.h"
#define ALEN(a,b,c,d) sqrt(((b)-(a))*((b)-(a))+((d)-(c))*((d)-(c)))
#define TEMPFACTOR 0.98

int metrop(float de, float t);
float revcst(float x[], float y[], int iorder[], int N, int n[]);
void reverse(int iorder[], int N, int n[]);
/*reverse move*/
float trncst(float x[], float y[], int iorder[], int N, int n[]);
void trnspt(int iorder[], int N, int n[]);
/*transport move*/

void anneal(float x[], float y[], int iorder[], int N){
/* This algorithm finds the shortest round-trip path to N cities
whose coordinates are in the arrays x[1..N],y[1..N]. The
array iorder[1..N] specifies the order in which the cities are
visited. On input, the elements of iorder may be set to any
permutation of the numbers 1 to N. This routine will return
the best alternative path it can find. */
int ans, L, sizefactor, initprob, alpha, minpercent, changes, i1, i2, tcount, fcount, trials, FLIMIT;
int counter = 0;
int i, j, k, nsucc, nn, idec;
static int n[7];
float path, de, t;
t = 0.90; /*temperature */
```

```
tcount = 1; /* Counter for temperatures*/
trials = 0; /* Counter for temperatures*/
fcount = 0; /* Counter for frozen*/
changes = 0; /* Counter for moves*/
FLIMIT = 5; /* Limit for frozen*/
sizefactor = 1000;
L = sizefactor * N; /* max number of paths at any temp */
initprob = 0.90; /* initial probability */
minpercent = 0.01; /* Breakout minpercent */
path = 0.0; /*initial path length*/
alpha = 0.05; /*alpha cutoff percentage */

printf("Initial order:\n"); /*print initial order */
for (i=1; i <= N; i++) /*calculate the distance of the TSP */
printf("city %d \n", iorder[i]);
for (i=1; i<N; i++) {
/* calculate initial path length */
i1 = iorder[i];
/*Determine position in the permuted solution space.*/
i2 = iorder[i+1];
path += ALEN(x[i1], x[i2], y[i1], y[i2]);}
i1 = iorder[N];
i2 = iorder[1];
path += ALEN(x[i1], x[i2], y[i1], y[i2]);
printf("path length %12.6f \n", path);
/*print path length*/

do {
changes = trials = 0;
do {
trials = trials + 1; /* try up to 100 temperature steps */
nsucc = 0;
for (k = 1; k <= L; k++) {
do {
n[1] = 1 + (int) (N * Random()); /*n[1] is start of the segment*/
n[2] = 1 + (int) ((N - 1) * Random()); /*n[2] is end of the segment*/
if (n[2] >= n[1]) ++n[2];
nn = 1 + ((n[1] - n[2] + N - 1) % N);} /*nn is number of cities not in segment*/
while (nn < 3);
idec = Equilikely(0, 1); /* decide between segment reversal and transport */
if (idec == 0) { /* do a transport */
n[3] = n[2] + (int) (abs(nn - 2) * Random()) + 1;
n[3] = 1 + ((n[3] - 1) % N); /* transport to a location NOT on the path */
de = trncst(x, y, iorder, N, n); /* find cost */
if (de <= 0.0) { /* a downhill move */
changes = changes + 1;
counter = 0; /* reset counter to 0 */}
else { /*an uphill move */
if (0.5 >= exp(-de / t)) { /*allow if true */
changes = changes + 1;
counter = 0;}}
ans = metrop(de, t);
if (ans) {
++nsucc;
path += de;
trnspt(iorder, N, n);}} /* transport */
else { /* do a path reversal */
de = revcst(x, y, iorder, N, n); /* de = change in objective function */
if (de <= 0.0) { /* a downhill move */
changes = changes + 1;
counter = 0; /* reset counter to 0 */}
else { /*an uphill move */
if (0.5 >= exp(-de / t)) { /*allow if true */
changes = changes + 1;
counter = 0;}}
ans = metrop(de, t);
if (ans) {
++nsucc;
path += de;
reverse(iorder, N, n);}} /* reversal */
if ((changes / trials) <= minpercent) {
fcount = fcount + 1;
printf("Fcount", fcount); } } /*break out of a higher temp */
printf("\n %s %10.6f :%s %12.6f \n", "T =", t, " Path Length =", path);
printf("Successful Moves: %6d\n", nsucc);
printf("T Count: %d", tcount);
t *= TEMPFACTOR;
fcount = 0;
tcount = tcount + 1;
if (nsucc == 0) return;} while ((trials < L) && (changes < (alpha*N))); }
while (fcount < FLIMIT);}
void reverse(int iorder[], int N, int n[]) {
int nn, j, k;
int m;
int itmp;
nn = (1 + ((n[2]-n[1]+N) % N))/2;
for (j=1; j<=nn; j++) {
k = 1 + ((n[1]+j-2) % N);
m = 1 + ((n[2] - j + N) % N);
itmp = iorder[k];
iorder[k] = iorder[m];
iorder[m] = itmp;}}
void trnspt(int iorder[], int N, int n[]) {
int m1, m2, m3, nn, j, jj, *jorder;
jorder = ivector(1, N);
m1 = 1 + ((n[2] - n[1] + N) % N);
m2 = 1 + ((n[5] - n[4] + N) % N);
m3 = 1 + ((n[3] - n[6] + N) % N);
nn = 1;
for (j=1; j <= m1; j++) {
jj = 1 + ((j + n[1] - 2) % N);
jorder[nn++] = iorder[jj];}
```

```
for (j=1; j <= m2; j++) {
jj = 1 + ((j + n[4] - 2) % N);
jorder[nn++] = iorder[jj];}
for (j=1; j <= m3; j++) {
jj = 1 + ((j + n[6] - 2) % N);
jorder[nn++] = iorder[jj];}
for (j=1; j <= N; j++)
iorder[j] = jorder[j];
free_ivector(jorder, 1, N);}
float trncst(float x[], float y[], int iorder[], int N, int n[]){
float xx[7], yy[7], de;
int j, ii;
n[4] = 1 + (n[3] % N);
n[5] = 1 + ((n[1] + N - 2) % N);
n[6] = 1 + (n[2] % N);
for (j = 1; j <= 6; j++){
ii = iorder[n[j]];
xx[j] = x[ii];
yy[j] = y[ii];}
de = -ALEN(xx[2], xx[6], yy[2], yy[6]);
de -= ALEN(xx[1], xx[5], yy[1], yy[5]);
de -= ALEN(xx[3], xx[4], yy[3], yy[4]);
de += ALEN(xx[1], xx[3], yy[1], yy[3]);
de += ALEN(xx[2], xx[4], yy[2], yy[4]);
de += ALEN(xx[5], xx[6], yy[5], yy[6]);
return de;}
float revcst(float x[], float y[], int iorder[], int N, int n[]) {
float xx[5], yy[5], de;
int j, ii;
n[3] = 1 + ((n[1] + N - 2) % N);
n[4] = 1 + (n[2] % N);
for (j = 1; j <= 4; j++){
ii = iorder[n[j]];
xx[j] = x[ii];
yy[j] = y[ii];}
de = -ALEN(xx[1], xx[3], yy[1], yy[3]);
de -= ALEN(xx[2], xx[4], yy[2], yy[4]);
de += ALEN(xx[1], xx[4], yy[1], yy[4]);
de += ALEN(xx[2], xx[3], yy[2], yy[3]);
return de;}
int metrop(float de, float t) {
return de < 0.0 || Random() < exp(-de/t);}
#define CITIES 10
int main (void) {
int i;
int N = CITIES;
int iorder[CITIES+1];
float x[CITIES+1], y[CITIES+1];
long SEED= 31764.0;

PlantSeeds(SEED);
```

```
for (i=1; i <= N; i++)
iorder[i] = i;

x[1]=1;
y[1]=3;
x[2]=2;
y[2]=2;
x[3]=5;
y[3]=9;
x[4]=8;
y[4]=8;
x[5]=2;
y[5]=7;
x[6]=7;
y[6]=1;
x[7]=5;
y[7]=5;
x[8]=6;
y[8]=4;
x[9]=3;
y[9]=4;
x[10]=1;
y[10]=5;
anneal(x,y,iorder,ncity);
printf("\nOrder after anneal:\n");
for (i=1; i <= ncity ; i++)
printf("city %d \n", iorder[i]);}
```

b.) *Test your altered code on the 10 city example. Compare the results with the results obtained from your experiments in part 2.*

For our altered code, 5 replications were run and the results recorded. The seeds that we used are as follows: $31764.0, 35711.13, 17192.32, 01123.58$, and 13213.45.

Table 6: Altered Code

Run	T Count	Final T	Final Path Length	Optimal Path
1	15	0.02288	28.4733	$(3, 5, 10, 1, 2, 9, 6, 8, 7, 4)$
2	8	0.04783	28.4733	$(1, 2, 9, 6, 8, 7, 4, 3, 5, 10)$
3	12	0.03138	28.4733	$(6, 8, 7, 4, 3, 5, 10, 1, 2, 9)$
4	11	0.03487	28.4733	$(10, 1, 2, 9, 6, 8, 7, 4, 3, 5)$
5	10	0.02288	28.4733	$(10, 1, 2, 9, 6, 8, 7, 4, 3, 5)$
Averages	**11.2**	**0.3197**	**28.4733**	

In part 2, we determined that the test parameters used in Trial 3 were the most efficient and effective. They were as follows:

Parameter	Setting
NOVER	= 100*NCITY
NLIMIT	= 10*NCITY
T	= 0.1
TFACTOR	= 0.90
N	= NCITY

The use of these parameters in our trial runs resulted in an average temperature count of 12.2 and an average final path length of 28.4733.

In our altered code, we used the following parameters:

Parameter	Setting
T	= 0.1
TFACTOR	= 0.90
SIZEFACTOR	= 100
INITPROB	= 0.40
MINPERCENT	= 0.20
ALPHA	= 0.10
FLIMIT	= 5
N	= NCITY

These parameters, coupled with our modifications to the code to reflect the standards in the Johnson et. al. paper, resulted in an average temperature count of 11.2, and an average final path length of 28.4733. The reduction in temperature counts corresponds to a reduction in required computer resources, thereby making the altered code more efficient. This efficiency maybe even greater than these runs suggest as there seems to be an outlier in the temperature count of run 1. More runs could provide a better estimate of the savings in temperature count. Note that the altered code still found the best minimum path length found in our trial runs, meaning that it is just as effective.

c.) *Change your code so that it can read an external file containing (x, y) coordinate data. There is coordinate information for 194 cities in the data file qatar.dat.*

The following alterations were made to the 'main' portion of our code:

```
int main (void) {
int i, j;
int N = CITIES;
int iorder[CITIES+1], node[CITIES+1];
float x[CITIES+1], y[CITIES+1];
FILE *fp;

long SEED= 13213.45;
/*keeping the same seed value allows you to duplicate an experiment.
The seed value can be anything. */
PlantSeeds(SEED);

for (i=1; i <= N; i++)
iorder[i] = i;
/*coordinates for original cities, to read in another file,
use an 'io' (input output) routine to read in a data file*/

for(i=1; i<= N; i++)
iorder[i] = N -i +1;

fp = fopen("qatar2.dat", "r");

if(fp == NULL) {
printf("\nError: Unable to open file.\n");
return;
}

for(j = 1; j<= N; j++){
if(fscanf(fp, "%i%f%f", &node[j],&x[j], &y[j]) != 3){
printf("\nError: Incorrect input.\n");
return;
}
}
anneal(x,y,iorder,N);

printf("\nOrder after anneal:\n");
for (i=1; i <= N ; i++)
printf("city %d \n", iorder[i]);
}
```

We then ran five initial test runs with the altered code and the parameters utilized in part **b.)**. The seeds that we used were as follows: $3174.0, 35711.13, 17192.32, 01123.58$, and 13213.45.

Table 7: Qatar Trial 1

Run	T Count	Final T	Final Path Length
1	12	0.031381	10491.218
2	11	0.034868	10054.217
3	10	0.038740	10231.823
4	12	0.031380	10265.784
5	14	0.025419	10118.305
Averages	**9.8**	**0.0323576**	**10250.269**

d.) *Retune your SA parameters and test the performance of your code for a suite of SEED values. Compare your results for the country of Qatar with the optimal tour at http://www.tsp.gatech.edu/world/countries.html.*

The website tells us that the optimal tour is a path length of 9352, which is less than our average final path length. This tells us that us that there is room for improvement in our code. We want to find methods for increasing our effectiveness. The Johnson et. al. paper describes in detail how the parameters are to be selected. Thus, we based our parameter selection on their parameter experiments. SIZEFACTOR was taken to be 10*NCITIES instead of neighborhood size. The parameters used in the Qatar test runs are as follows:

Parameter	Setting
T	= 0.40
TFACTOR	= 0.98
SIZEFACTOR	= 1950
INITPROB	= 0.90
MINPERCENT	= 0.01
ALPHA	= 0.05

Table 8: Qatar Trial 2

Run	T Count	Final T	Final Path Length
1	42	0.174715	10161.982
2	51	0.145668	9845.904
3	50	0.148641	10039.728
4	88	0.068981	9772.328
5	128	0.030745	9704.330
Averages	**71.8**	**0.11375**	**9905.774**

Fine tuning the parameters has indeed resulted in a better average path length of 9905.774 in Trial 2 versus 10250.269 in Trial 1. This was done, however, at a cost of decreased efficiency as the average T count increased from 9.8 to 71.8. Though our final path length was shorter, we were not able to detect the optimal path length of 9352. We can therefore assume that there are either better parameters to employ or, perhaps, a better starting solution. Johnson et. al. details methods for selecting a better starting solution. Selecting a better starting solution may increase our chances of finding the optimal path.