CHAPTER 5
COMPUTATIONAL PERFORMANCE TESTS FOR ALGORITHM TD-*k*SP

5.1. List of Tests Carried Out

Algorithm TD-*k*SP was tested for various network sizes. In particular we study the variation in computational time with respect to changes in the following network characteristics.

1. Number of nodes in the network.
2. Number of arcs in the network.
3. Density of the network.

In addition to these tests, we study the variation in computational time with respect to the following shortest path parameters.

1. The value of '*k*'.
2. The number of starting times (*M*).

The tests were conducted on a SUN-SPARC 1000 computer. The algorithms were coded using the *C* programming language along with some specialized data structures. The algorithm was run for various parameter values and the computational times for each run was tabulated. Extensive statistical tests were then run to validate results. There has been no other implementation scheme reported in the literature that uses these two types of random networks in tandem to test the efficiency of TD-*k*SP algorithms.

5.2. The Random Network Generator Programs

5.2.1. The Random Connected Digraph

Problem Statement.

Generate a digraph having *m* nodes and a density ρ, such that the arcs in the network are of random length and connection (*i*.e., we should not be able to predict with 100%

certainty the existence or the length of an arc between any two given nodes in the generated network) and such that at least one path exists between the origin and all other nodes in the network. Note that the density of a network $G(N, A)$ is defined as ratio of the number of arcs in the network to the number of arcs in a dense (fully connected) network having the same number of nodes. The number of arcs in a dense network can be computed using the combinatorial expression ${}^{m}P_{2}$, *i*.e., the maximum possible permutations (arrangements) of m objects, taking two of them at a time. This value is equal to $m(m\text{-}1)$ for digraphs. Using the definition for density, we can obtain the desired number of arcs to be equal to $\lceil \rho m(m\text{-}1) \rceil$. The problem can be solved (in polynomial time) using the following scheme.

1. Generate random edge lengths L_{ij} for a dense network having m nodes.

2. Find the minimum spanning tree for this dense network, rooted at the origin node. This can be achieved by an efficient implementation of the (static) LC or LS shortest path algorithm. Note that the fully dense network corresponds to the maximum possible network size for a given number of nodes. Usually, the computational time complexity is of the order $O(m^2)$ for an LC algorithm or $O(m \log m)$ for an LS algorithm. Hence, for large values of m, the computational times for $O(m^2)$ algorithms increases tremendously (but not exponentially). Obviously, an $O(m \log m)$ LS algorithm would perform much better for large values of m, whereas for small values of ρ and/or m, the LC algorithm might be a better choice. In fact, the random network generator can itself be used to compare performances of LC and LS algorithms.

The minimum spanning tree generated is a digraph having m nodes and $(m\text{-}1)$ arcs, with each of the nodes being accessible to the origin via an unique path. Until now we have fulfilled the criterion of connectedness for the network. Now the task remains for us to (randomly) generate the remaining $\lceil \rho m(m\text{-}1) \rceil - (m\text{-}1)$ arcs in the network.

3. To generate the remaining arcs, we use a (pseudo) random number generator.

We first calculate the density d, of the residual network as shown below, noting that the maximum number of arcs available for addition is now $m(m\text{-}1) - (m\text{-}1) = (m\text{-}1)^2$. Setting

$$d(m\text{-}1)^2 + (m\text{-}1) = \rho m(m\text{-}1),$$

we obtain

$$d = (\rho m\text{-}1) / (m\text{-}1).$$

Hence, we generate $(m\text{-}1)^2$ uniformly distributed random numbers (u_i) between [0,1]. Compare each u_i with a threshold value of d. If $u_i < d$, generate a link having a length L_{ij} between the head node and tail node associated with link i. On the average, a network having a density of d will be generated if the random number generator is well-behaved. Note that the network has full connectedness and therefore it is a good basis to compare the performance of LC and LS algorithms. Note that if there are some nodes that are inaccessible from the origin, their labels would remain unaltered, while being considered a potential candidate for future (LS) updates. This anomaly may bias the results favorably towards algorithms that use an SEL (LC methods) when compared to LS algorithms, which have to store the entire set of nodes with temporary labels.

5.2.2. **The Random Layered Network**

Another useful test graph that can be generated to test k-SP algorithms is the layered random acyclic network. Here, the link lengths are random but the structure and density of the network can be user-defined. The characteristics of a layered network can be specified by the 2-tuple (p, r), where r represents the out-degree of a node, and the number of layers in the network is (p+2). The origin O is situated at layer 0, and the destination D is located at layer (p+1), and all the nodes at layer p are individually connected to the destination node D. This (acyclic) network has an extremely large number of O-D paths (r^p), and therefore, one can measure the efficiency and verify the (strongly) polynomial-time behavior of the LC or LS algorithm by increasing the number of O-D paths and studying the corresponding increase in the computational time. Note that now, the algorithm will now have to select the k best O-D paths from an exponential number of such paths.

Consider a layered network with the 2-tuple (p, r) associated with it. We can calculate the following network parameters.

Number of nodes in the network (m) = $2 + r(r^p\text{-}1) / (r\text{-}1)$.

Number of arcs in the network (n) = $r^p + r(r^p\text{-}1) / (r\text{-}1)$.

As the value of p increases (keeping r fixed), the number of *O-D* paths increase exponentially in p, while the number of paths to any other intermediate node is always limited to one.

5.3. **Computational Results for Algorithm TD-*k*SP**

The algorithm was tested for various network sizes and densities, and for various values of k and starting times. The algorithm was also tested for specially constructed acyclic layered networks. The CPU processing time (in seconds) of the computer (T) was tabulated for each problem instance. Table 6 illustrates the results for various sizes of randomly generated networks for a value of k = 10. In general, the networks in Table 6 correspond to sparse networks (networks having low density). An empirical expression for T is derived. These results are shown in Table 7. Similarly, Table 8 illustrates the computational results for networks having a large number of arcs (the density may correspond to sparse or non-sparse networks). The variation of computational time with an increase in the value of k, along with empirical equation fits, are illustrated in Tables 9 and 10. A similar test with increasing values of starting times (M) are shown in Tables 11 and 12. The computational performance characteristics of the algorithm for random layered networks in presented in Tables 13, 14 and 15. In addition, some key observations are made, based on computational experience, regarding the performance of the TD-*k*SP algorithm (for which no computational results are shown). Empirically, the algorithmic behavior resembles an "almost linear" $O(n^x)$ or a $O(m\rho^x)$ algorithm, where $x \approx 1.0$ - 1.5. Finally a multiple regression model is used to derive an empirical expression for the time-complexity of the algorithm. These results are shown in Tables 16-18.

Table 6. Computational Times (in Seconds) for Extremely Sparse Random Networks

m = 100

n	100	200	300	400	500	600	700	800	900	1000
ρ (%)	1.01	2.02	3.03	4.04	5.05	6.06	7.07	8.08	9.09	10.10
T	0.09	0.23	0.40	0.46	0.93	1.08	1.23	1.58	1.73	2.15

m = 200

n	200	300	400	500	600	700	800	900	1000
ρ (%)	0.50	0.75	1.01	1.26	1.51	1.76	2.01	2.26	2.51
T	0.15	0.31	0.65	0.79	0.90	0.97	1.17	1.20	2.33

m = 300

n	300	400	500	600	700	800	900	1000
ρ (%)	0.33	0.45	0.56	0.67	0.78	0.89	1.00	1.15
T	0.29	0.42	0.60	0.78	0.97	1.12	1.19	1.41

m = 400

n	400	500	600	700	800	900	1000
ρ (%)	0.25	0.31	0.38	0.44	0.50	0.56	0.63
T	0.47	0.58	0.71	0.94	1.03	1.14	1.35

The computational times T obtained were tabulated for various (percentage) density values (ρ (%)). Table 6 illustrates the effect of the density of the network (or arc-to-node ratio) on computational times. To observe this feature consider the table shown above to be a upper-triangular matrix of results. As we move across any row (increasing the number of arcs, while holding the number of nodes at a constant level), there is an increase in computational time with an increase in density, as expected. A more surprising observation can be made my moving down any column (holding the number of arcs constant and increasing the number of nodes). Here, as the size of the network increases, the computational time actually decreases. Such a result is unlikely for LS algorithms. To explain this result, let us make a second observation. As we move down a column, the network density (or arc-to-node ratio) decreases, while the number of arcs remain constant. These observations seem to indicate that the algorithm is primarily influenced by the density of the network for small values of ρ. Now, if we look diagonally across the matrix, holding the arc-to-node ratio (μ) constant and increasing the network size (number of nodes), we observe an (almost proportional) increase in computational time. Initially, for m = 100, an empirical expression for T can be obtained as a function of the number of arcs as $T \propto n^{1.4}$. However, the exponent for n in this expression increases as m increases. These results indicate that the density of the network is not the sole factor affecting computational times (which is to be expected). On the basis of these results we can tentatively postulate that the expression for computational time T can be written as

$T \propto m^{\alpha}\rho^{\beta}$ or

$\log (T) = c + \alpha \log (m) + \beta \log (\rho)$,

where c is a constant. Linear (or multiple) regression methods can used to determine the parameters (α, β). In particular, we can use a linear regression model by setting $\alpha = 1$ and holding m constant. We can perform these calculations for various values of m and tabulate these results. The expression for T was obtained for the values of m shown in Table 1 using a log-log fit. The expressions are listed below in Table 7. The coefficient of correlation γ, is indicated in each case.

Table 7. Empirical Expressions for $T(\rho)$ and $T(m)$

m	γ	**log (*T*)**
100	0.995	-1.065 + 1.377 log (100ρ)
200	0.971	-0.317 + 1.456 log (100ρ)
300	0.996	0.097 + 1.301 log (100ρ)
400	0.996	0.358 + 1.154 log (100ρ)

ρ (%)	γ	**log (*T*)**
1.0	0.996	-3.53 + 1.25 log (*m*)
0.75	0.999	-3.645 + 1.271 log (*m*)
0.5	0.993	-4.473 + 1.575 log (*m*)

The empirical results using the log-log fit indicate that we can use an approximate value of 1.3 for α and β (only for sparse networks). These initial values of α and β can then be fine-tuned to obtain an "exact" expression for the computational time of the algorithm. We can also experiment with smaller values of α and β (or use multiple regression) to obtain better fits. The next set of results demonstrate the extremely fast performance of graph-theoretical methods such as Algorithm TD-*k*SP.

Computational times are listed below for networks with one thousand or more links. It must be mentioned here that generating large random connected networks is an extremely tedious procedure. A procedure to generate a network having 500 nodes and 1000 arcs takes approximately two days, and a similar procedure to generate a network having 1000 nodes and 2000 arcs takes 14 days[1]. Computations involving larger network sizes imposed a heavy burden on the computer resources that were available and could not be generated. However, a real-life transportation network for the city of Seminole, Florida

[1] Thus demonstrating that even an $O(m^2)$ algorithm can be expensive for large values of m. Usually, an O(n log m) algorithm will perform much better for large values of m.

was obtained. This network contains more than 3000 nodes and 4000 arcs. The algorithm was tested for this network and performed extremely well (see Table 3). In general, LC algorithms can efficiently solve SP problems on extremely sparse networks. Typically, road networks (obtained from digitized GIS coverage files) are extremely sparse ($\mu < 3$), having many disjoint components, and LC algorithms will most probably perform well on such networks. Similarly, LS algorithms will perform better if the network has a denser structure and connectivity. However, the path processing data structure (PPDS) used is a crucial factor. In the random network generator programs, the LS algorithm out-performed the LC algorithm for sparse networks when a linked-list was chosen as the PPDS.

Table 8. Computational Times (in seconds) for Large Random Networks

$m = 100$

n	1000	2000	3000	4000	5000	10000
T	2.15	4.09	5.74	7.40	8.96	17.49

$m = 200$

n	1000	2000	3000	4000	5000	10000
T	2.33	5.44	6.55	8.39	10.12	19.56

$m = 300$

n	1000	2000	3000	4000	5000	10000
T	1.41	4.85	7.09	9.33	11.71	20.83

$m = 400$

n	1000	2000	3000	4000	5000	10000
T	1.35	4.35	6.43	9.03	11.22	22.60

$m = 500$

n	1000	2000	3000	4000	5000	10000
T	1.72	3.32	7.76	10.08	14.52	26.07

m	***n***	**Network Type**	***T***
1000	2000	random	25.34
3304	4526	real-life	4.36

A quick inspection of the results in Table 8 show that the computational times increase almost linearly with an increase in the number of arcs. Hence, another expression can be obtained for T as a function of n and m. Such a procedure would be similar to the one used in Table 7.

Until now, we have considered the problem of finding a single shortest path. In many cases, finding the k best paths becomes important. In constrained shortest path problems, k shortest path algorithms can be used to obtain the k shortest paths (that have the k lowest objective values). These k paths can then be processed to find the path that best satisfies additional side constraints. One of the most important criteria for k shortest paths is speed. The TD-kSP algorithm was tested for various values of k on large random networks having 400 nodes and for different densities. The values of k are made to increase exponentially, and the corresponding values of T are tabulated. These results are illustrated in Table 9.

Table 9. Computational Times (in seconds) for Varying values of k

$m = 400$, $n = 1000$

$\boldsymbol{k}$	1	2	4	8	16	32	64	128	256
$\boldsymbol{T}$	0.77	0.77	0.85	1.11	2.65	12.05	76.72	503.27	1082.76

$m = 400$, $n = 2000$

$\boldsymbol{k}$	1	2	4	8	16	32	64	128
$\boldsymbol{T}$	1.46	1.51	1.72	3.03	12.59	77.57	506.27	931.91

$m = 400$, $n = 3000$

$\boldsymbol{k}$	1	2	4	8	16	32	64	128
$\boldsymbol{T}$	2.16	2.23	2.58	4.53	17.49	106.81	730.94	

$m = 400$, $n = 4000$

$\boldsymbol{k}$	1	2	4	8	16	32	64	128
$\boldsymbol{T}$	5.07	2.93	3.47	8.38	25.67	155.13	1043.32	1957.57

$m = 400$, $n = 5000$

$\boldsymbol{k}$	1	2	4	8	16	32	64	128
$\boldsymbol{T}$	3.73	3.68	4.58	7.91	32.06	198.49	1327.01	

The empirical results indicate that the computational time increases at a rate proportional to (approximately) $k^{1.44}$ for an unit increase in k. The reason for this non-linearity is that at each label-correcting step, as we update labels of nodes, $O(k)$ comparisons need to be performed (in the worst case) to obtain the k vector. These computations need to be performed k times, to yield an overall worst-case time-complexity of $O(k^2)$. Note that the empirical value of $k^{1.44}$ represents an "average case" complexity. Compare the expression for T in Table 10 (shown below) with the results in Table 7, where an empirical expression for T is obtained as a function of the M (the number of starting times). Tables 11 and 12 show that we can calculate the SPs for each starting time much faster than computing the SPs for the same value of k. One way of reducing the exponent value from 1.44 to a smaller value is to use efficient data structures. For example, a binary heap can be used instead of the linked-list to improve the time-complexity to $O(k \log_2 k)$.

Table 10. Empirical Expressions for $T(k)$

n	γ	**log (*T*)**
1000	0.942	-0.746 + 1.440 log (*k*)
2000	0.950	-0.360 + 1.497 log (*k*)
3000	0.924	-0.136 + 1.398 log (*k*)
4000	0.938	0.056 + 1.436 log (*k*)
5000	0.926	0.093 + 1.419 log (*k*)

Table 11. Computational Times (in Seconds) for Varying Values of *M*

m = 400, *n* = 1000

M	1	2	4	8	16	32	64	128	256	512	1024
T	0.72	0.75	0.77	0.80	0.87	1.01	1.27	1.83	2.89	5.09	9.52

m = 400, *n* = 2000

M	1	2	4	8	16	32	64	128	256	512	1024
T	1.44	1.44	1.46	1.55	1.64	1.87	2.35	3.28	5.14	8.85	16.34

m = 400, *n* = 3000

M	1	2	4	8	16	32	64	128	256	512	1024
T	2.13	2.15	2.20	2.28	2.45	2.71	3.38	4.61	7.18	12.29	22.78

m = 400, *n* = 4000

M	1	2	4	8	16	32	64	128	256	512	1024
T	2.88	2.90	2.94	3.01	3.26	3.64	4.45	6.06	9.39	16.00	29.09

m = 400, *n* = 5000

M	1	2	4	8	16	32	64	128	256	512	1024
T	3.64	3.61	3.65	3.81	4.02	4.51	5.48	7.43	11.38	19.41	35.32

The results from Table 12 show that the algorithm computes the 10 best shortest paths to all nodes in the network for more than 1000 starting times within a CPU time of 36 seconds for even the largest network size. A empirical relation between T and M was obtained. As expected, an almost exact linear fit was obtained. A comparison with the results in Table 12 indicates that in come instances, the algorithm is more than 200 times faster for some values of M, when compared to the results in Table 11 for $k = M$. However, in practice, the value of k is usually small compared to M. As a result the relatively poor performance of Algorithm TD-kSP will not have a drastic impact in such situations.

Table 12. Empirical Expressions for $T(M)$

n	**γ**	***T***
1000	1.0	0.725 + 8.572 (M/1000)
2000	1.0	1.413 + 14.567 (M/1000)
3000	1.0	2.085 + 20.138 (M/1000)
4000	1.0	2.824 + 25.660 (M/1000)
5000	1.0	3.523 + 31.023 (M/1000)

Until now we have considered connected random networks. In some shortest path applications, acyclic networks are encountered. The next few sets of results illustrate the computational efficiency of the algorithm when applied to such acyclic, layered test networks.

Table 13. Computational Times for Layered Networks

$r = 2, k = 20$

p	2	3	4	5	6	7	8	9	10	11
***O-D* paths**	4	8	16	32	64	128	256	512	1024	2048
m	8	16	32	64	128	256	512	1024	2048	4096
n	10	22	46	94	190	382	766	1534	3070	6142
T	0.02	0.03	0.05	0.09	0.21	0.49	1.27	3.91	13.40	49.41

$r = 3, k = 20$

p	2	3	4	5	6	7	8
***O-D* Paths**	9	27	81	243	729	2187	6561
m	14	41	122	365	1094	3281	9842
n	21	66	201	606	1821	5466	16401
T	0.03	0.07	0.22	0.83	4.82	35.24	308.72

$r = 4, k = 20$

p	2	3	4	5	6
***O-D* paths**	16	64	256	1024	4096
m	22	86	342	1366	5462
n	36	148	596	2388	9556
T	0.04	0.14	0.80	7.22	98.48

$p = 4, k = 20$

r	2	3	4	5	6	7	8	9	10
***O-D* paths**	16	81	256	625	1296	2401	4096	6561	10000
m	32	122	342	782	1556	2802	4682	7382	11112
n	46	201	596	1405	2850	5201	8776	13941	21110
T	0.06	0.22	0.80	2.83	9.53	28.82	129.08	217.47	423.01

The results in Table 13 exhibit the variation in network parameters and computational time with an increase in the number of layers (p). Also, the variation in computational time with the out-degree (r) of a node is also shown as a comparative statistic. Tentatively, let us hypothesize that $T = T(m)$. The log-log fits for various values of p are shown in Table14.

Table 14. Empirical Expressions for $T(m)$

r	γ	**log (*T*)**
2	0.985	-3.139 + 1.256 log (m)
3	0.988	-3.438 + 1.413 log (m)
4	0.991	-3.498 + 1.418 log (m)

From the results of Table 14, we see that the empirical exponents for m marginally increase with an increase in p. Hence, it is apparent that the computational time is a multivariate function. We notice that as p increases the arc-to-node ratio (μ) also increases. Hence, we can postulate that the computational time T can be expressed as

$$T \propto m\mu^{x}$$

where x is the exponent for α to be empirically determined. We can expect x to be close to unity. Using a process of trial-and-error, let us use a value of $x = 1.3$.

Table 15. Empirical Expressions for $T(m, \alpha)$

r	γ	**log (*T*)**
2	0.981	-27.082 + 0.016 $m\mu^{1.3}$
3	0.976	-15.556 + 0.016 $m\mu^{1.3}$
4	0.974	-5.558 + 0.009 $m\mu^{1.3}$

We can see that except for $r = 4$, a value of 1.3 for x seems to be good. Note that for $r = 4$, a good fit was obtained for $T(m)$. Hence, a lower value of x may have been better for this case. A value of $x = 1.1$ or 1.2 may yield better overall results. Based on these

results, $T \propto m\mu^{1.2}$ can be tentatively used for layered networks. However, a more accurate analysis of these results can be performed using a multiple (logarithmic) regression model. These tests are described next.

5.4. **Multiple Regression Analysis**

The computational performance results for the algorithm was statistically analyzed using the freeware (demo version) *NCSS-JR 6.0*[2]. In particular, a multiple regression model

$$Y = B_0 + B_1X_1 + B_2X_2 + ..B_pX_p$$

where,

the vector $\{B\}$ represents the set of regression coefficients that are to be ascertained using a least-squares fit ($B_o \equiv c$, $B_1 \equiv \alpha$, $B_2 \equiv \beta$),

Y represents the dependent variable ($\log T$),

and the vector $\{X\}$ represents the set of independent variables ($\log m$, $\log 100\rho$).

Note that until now, we tried to fit equations with $p = 1$ for various X variables.

This model was used to obtain an empirical relationship between T, m and ρ. All the computational results presented in Tables 6-13 were used as input for the model. The regression model was run for the three cases shown below.

1) $\rho < 10\%$.
2) ρ unrestricted.
3) Layered Networks.

Standard statistical tests were carried out by the software and graphical plots can be obtained for these test results. The confidence level of the regression-fit, correlation coefficients, error estimates, etc., and other useful statistical estimates were also recorded. The results for three cases are presented in Tables 16, 17, and 18 respectively. Complete statistical results is presented in the appendix. Tables 16-18 show the statistical results for

[2] Available for free download from the Internet web-site: http://www.ncss.com/donwload.html

the regression equation and regression coefficients. In all three cases, the results indicate a high coefficient of correlation (R^2) coefficient value of 0.98, which is close to 1. The empirical regression coefficients were validated for a 95 % confidence level (C.L). The main assumption for these tests is that the residuals (the difference in the actual and predicted computational times) are normally distributed and variance is constant. The normality and variance test results conducted justify this assumption. Selective definitions that are useful in interpreting these tables are presented in the appendix. For a detailed description of such tests, the reader is referred to user's guide of the software package. Complete results are shown only for the first case (random networks, $\rho < 10\%$). For the other two cases, only the tests for the coefficients obtained are shown.

For the first case, the empirical equation obtained for T is

$T \propto m^{2.4}\rho^{1.2}$ or $T \propto n^{1.4}\mu^{1.2}$

where μ (arc-to-node ratio) is substituted using the approximate equation

$\mu = m\rho$.

Similarly, for the second and third cases we obtain $T \propto n^{1.5}\mu^{1.4}$ and $T \propto n^{1.5}\mu^{1.1}$, respectively. A comparison of these results with the linear regression (single variable) confirm the trends obtained in Tables 6-13. A comparison of the results in Table 11 and Table 12 show that the algorithm performs relatively poorly for denser networks. The results for layered networks (Table 13) show an almost linear variation of computation time with density. Interestingly, the null hypothesis (no correlation with dependent variables) for the case of layered networks is accepted at the given significance level indicating that a linear fit may yield better results. Using the expressions obtained from Tables 1-13, we can postulate that the empirical time-complexity for Algorithm TD-*k*SP is of the order $O(k^{1.4}M m^{12} \mu^{1.2})$ or equivalently, $O(k^{1.4}M m^{2.4} \rho^{1.2})$. The exponents for n and μ in this expression may increase by 0.1 or 0.2 for dense networks ($\rho > 10\%$) and possible decrease for extremely sparse networks ($\rho < 0.1\%$). Further, in the case of acyclic networks, the algorithm seems to perform in a "almost" linear fashion.

Ziliaskopoulos and Mahmassani [1994] used a CRAY Y/MP-8 supercomputer with parallel processing abilities to test the TDSP algorithm they devised for

DYNASMART. They obtain an empirical time-complexity of the order $O(Mm\mu^{1.4})$ or equivalently, $O(Mm^{2.4}\rho^{1.4})$ for the case of $k = 1$ and $M < 640$ for extremely sparse networks of similar magnitudes of density ($\rho < 3$ %) that are used in this study. Based on empirical results, and noting that both algorithms essentially use the same PPDS (deque), Algorithm TD-kSP seems to perform marginally better for (increasingly) denser networks, while performing almost equally well for sparse networks. *DYNASMART* imposes the consistency assumption on link-delays for their tests (which is naturally true in our case). Hence, the algorithms must yield identical results, when they are used to find the shortest path for a given O-D pair. Based on these observations, we can conclude that when simple optimal paths are desirable, Algorithm TD-kSP can used to obtain the k shortest paths as efficiently as the currently existing algorithm. However, conclusive evidence to confirm these statements can be obtained only if Algorithm TD-*k*SP is also tested for larger networks ($m > 1000$).

Table 16. Multiple Regression Results for Random Networks and ρ < 10%

Regression Equation Tests

Independent Variable	**Regression Coefficient**	**Standard Error**	**T-Value (H_0: B = 0)**	**Prob. Level**	**Decision (5%)**	**Power (5%)**
c	-5.859053	0.1667649	-35.1336	0.0	Reject H_o	1.0
log *m*	2.418785	6.786013E-02	35.6437	0.0	Reject H_o	1.0
log 100ρ	1.240923	2.684141E-02	46.2317	0.0	Reject H_o	1.0

R^2 = 0.977257

Regression Coefficient Tests

Independent Variable	**Regression Coefficient**	**Standard Error**	**Lower 95% CL**	**Upper 95% CL**
c	-5.859053	0.1667649	-6.19401	-5.524096
log *m*	2.418785	6.786013E-02	2.282484	2.555086
log 100ρ	1.240923	2.684141E-02	1.18701	1.294835
T-Critical	2.008559	-	-	-

Table 16. cont'd.

Analysis of Variance (ANOVA) Results

Source	DF	Sum of Squares	Mean Square	F-Ratio	Prob. Level	Power (5%)
c	1	3.777783	3.777783	-	-	-
Model	2	14.27011	7.135055	1074.26	0.0	1.0
Error	50	0.3320912	6.641824E-03	-	-	-
Total	52	14.6022	0.2808115	-	-	-

Root Mean Square Error	8.149739E-02
Mean of Dependent Variables	0.2669811
Coefficient of Variation	0.3052552
PRESS Value	0.3890393

Normality Tests

Assumption	Decision (5%)
Skewness	Accepted
Kurtosis	Accepted
Omnibus	Accepted

Table 17. Multiple Regression Results for Random Networks and Unrestricted ρ

Regression Equation Tests

Independent Variable	**Regression Coefficient**	**Standard Error**	**T-Value (H_0: B = 0)**	**Prob Level**	**Decision (5%)**	**Power (5%)**
c	-6.039011	0.12363	-48.8475	0.0	Reject H_o	1.0
log *m*	2.48936	5.027575E-02	49.5141	0.0	Reject H_o	1.0
log 100ρ	1.362969	2.596588E-02	52.4908	0.0	Reject H_o	1.0

R^2 = 0.987129

Regression Coefficient Results

Independent Variable	**Regression Coefficient**	**Standard Error**	**Lower 95% C.L.**	**Upper 95% C.L.**
c	-6.039011	0.12363	-6.288507	-5.789516
log *m*	2.48936	5.027575E-02	2.387899	2.59082
log 100ρ	1.362969	2.596588E-02	1.310568	1.41537

Table 18. Multiple Regression Results for Layered Networks

Regression Equation Tests

Independent Variable	**Regression Coefficient**	**Standard Error**	**T-Value (H_0: B = 0)**	**Prob Level**	**Decision (5%)**	**Power (5%)**
c	-5.949147	2.813731	-2.1143	0.043523	Reject H_o	0.532539
log *m*	2.543608	1.274068	1.9964	0.055692	Accept H_o	0.487224
log 100ρ	1.146678	1.302185	0.8806	0.386041	Accept H_o	0.136123

R^2 = 0.973502

Regression Coefficient Tests

Independent Variable	**Regression Coefficient**	**Standard Error**	**Lower 95% C.L.**	**Upper 95% C.L.**
c	-5.949147	2.813731	-11.71281	-0.1854801
log *m*	2.543608	1.274068	-6.620239E-02	5.153418
log 100ρ	1.146678	1.302185	-1.520726	3.814083

We have finally reached a stage where the development of an interactive environment for deployment of optimization algorithms such as TD-*k*SP for real-time applications is complete. This concludes our work on the development, testing and implementation of TDSP algorithms. The objectives of this study, i.e., the formulation, analysis, computational testing, and implementation of dynamic routing algorithms, is now complete. In conclusion, the next section presents a summary of important results obtained. The future scope of this study and possible extensions to the work done is also described.