A Graph Based Methodology for the Representation and Evaluation of Text Input Strategies for Miniature and Mobile Devices

Frode Eika Sandnes
Dept. Computer Science
Faculty of Engineering, Oslo University College
P.O. Box 4 St. Olav's plass, N-0130 Oslo, Norway
frodes@iu.hio.no

Abstract–*In this paper a new methodology for representing text-input strategies for miniature and mobile devices is presented. The methodology is based on representing text-input strategies as graphs. Graph representations allow different static mobile text-input strategies to be represented in a uniform manner. Further, different strategies are easily compared as the graph representation allows various characteristics to be extracted. The methodology incorporates KSPC (KeyStrokes Per Character), checking for error recoverability and correctness. We also propose an error recovery measure – the mean error recovery distance (MERD). The methodology can be expanded to include additional evaluation measures and it is feasible to implement design-tool support. Finally, the methodology is demonstrated on several text entry designs from the literature.*

Keywords: mobile text entry, model, graph.

1. Introduction to mobile text entry

Mobile text input is an important aspect of contemporary human computer interaction. Users send SMS messages using mobile phones, edit song titles on miniature mp3 players and edit address lists on wristwatches, or portable databanks.

Small and portable devices are attractive to users. However, the small size results in small displays and less room for interaction controls. The limited surface area only allows for a few keys. There is also a trade-off between the number of keys and the size of the keys. Smaller keys are harder to use than larger keys and leads to higher error rates as incorrect keys are more easily pressed accidentally (Fitts' Law [1]). It is rarely room for full size keyboards, and mobile devices usually have fewer keys than there are characters in the alphabet.

For example, mobile phones allow users to hit a key repeatedly to cycle through characters labelled on the key to retrieve a desired character. A character is thus retrieved with anything from 1 to 5 keystrokes.

Two and three key text entry systems first appeared on arcade game machines in the 70'ties, where users employ rotator keys and press a select button to select the desired character. MacKenzie [5] describes the date-stamp approach at great detail.

Raghunath and Narayanaswami [6] implemented a wristwatch system, consisted of splitting the alphabet into two and presenting the alphabet as two rings. One key is used to toggle between the two rings, one key is used to cycle forward in the rings and the final key is used to select a character.

Sandnes et al. [7] investigated the Multi-ring where characters are organised into groups, for example 'abc', 'def', 'ghi' and so forth. First, the user cycles through the list of groups using a 'left' and a 'right' key and then selects the group containing the desired character. Then, the user cycles through the characters within the group.

Four key text entry strategies has been studied by Evreinova et al [8] and Tamaka-Ishii et al [9] and studies addressing five key text entry strategies include Moyes [10], Isokoski and Raisamo [11] and Bellman [12].

1.1. Benefits of a graph based methodology

There are several benefits to introducing a graph based methodology for modelling text entry strategies. First, the text entry strategies must be static in nature since graphs are static structures. Hence, no dynamic aspects can be modelled. There are several major challenges associated with dynamic and adaptive user interfaces from a usability point of view. Several studies on mobile text entry conclude that adaptive and dynamic text entry strategies are hard to learn and thus slow to use in practice although they are theoretically fast to use (See [5, 12]). Further, adaptive and dynamic strategies require feedback and cannot be used eyes-free.

Second, graph theory is well understood and widely used in computer science and a wide range of graph theoretic algorithms and graph analysis metrics exists, which can be applied to the evaluation and analysis of text-entry systems.

Third, the graph notation allows automated verification and analysis of text-entry designs to be carried out by automatic tools.

2. A graph based methodology

The purpose of this methodology is to identify weaknesses in text entry designs before one is committing to user tests, as testing on real people is an expensive and time-consuming activity. Note however that the methodology is not a substitute for testing with real users.

2.1. Scope

The methodology in this paper has two limitations: it applies to static text entry strategies where there is a limited set of control signals. A static text entry strategy can be defined as one which does not change throughout its lifetime and that its response is predictable. For example, text entry prediction and disambiguation both employ dictionaries to enhance and simplify the text entry process. In prediction the text entry system attempts to predict which word the user types based on the characters already entered [13], and it is an established technique within the realm of disabled users. Disambiguation-based techniques are often associated with the T9 system. Both prediction and disambiguation are difficult to capture in a static model and are therefore beyond the scope of this paper.

A limited set of control signals means that the physical characteristics of a device has only a small number of ways in which users can interact with the device. For example, a desktop has a full size QWERTY keyboard with more than 100 keys, while a wristwatch may only have three keys for interaction. Due to the limited number of control signals, several signals must therefore be combined into a sequence in order for symbols to be produced or operations to be executed. Interaction strategies which enjoy an unrestricted number of interaction signals has no need for sequential input and are therefore not relevant for this strategy.

We define text entry strategies where text is typed in one step as *simultaneous text entry*, for example standard QWERTY typing and chord typing. Further, text entry strategies were text is entered in multiple steps is termed *sequential text entry*.

2.2. Devices

In this study it is assumed that the device is of limited physical size and thus has a limited surface area. Further, it is assumed that the device has a form of visual, aural or tactile feedback and some interaction controls, such as keys. The interaction controls can be used to send interaction signals defined by the set $S = s_1, s_2, .., s_N$, where N is the total number of control signals. These controls are typically keys, and the devices would typically be one-hand devices, i.e. they are held in one hand and operated by the other hand. Note that the model is capable of representing two-handed operation as well. In this study devices with three keys are used for illustrative purposes. The concept of three buttons is appealing and it is the minimum number of buttons one need in order to practically operate a device and they can be operated with one hand only, in such a way that each key is assigned a unique finger. Implications of this is that the fingers do not need to be moved across a keyboard from one key to another and physical navigation errors and the delay associated with moving fingers between keys are eliminated [14]. Further, since the fingers constantly cover the keys, no backlighting of the keys is needed resulting in an overall reduced power consumption. However, the model proposed in this paper is not limited to just three keys.

2.3. Signals

Given a device with K keys, in this instance three keys, several categories of signals can be sent. First, *single keystrokes* can be used to send K different signals. Second, *chording* can be used to increase the number of possible signals. With K keys $K^2\text{-}1$ signals can be sent and with three keys 7 different signals can be chorded. Third, *keystroke duration* can be used to express different signals – for example short taps and long or hold strokes. With K keys $2K$ signals can be sent when allowing for short and long taps. If chording and keystroke duration is combined a total of $2(K^2\text{-}1)$ signals can be generated. For 3 key devices this would account to 14 different signals. Note however that a combination of chording and keystroke duration is difficult and would require practice.

In the proposed methodology single keystrokes are denoted by e_i where e_i indicates that key e_i is pressed, e_i+e_j indicate a chord signal comprising of the keys e_i and e_j, and a short tap is denoted by $\acute{e}_i$ and a long tap is denoted by $\hat{e}_i$.

2.4. The alphabet

The alphabet is a set of symbols that that the user needs in order to compose the desired texts. This study is restricted to languages using phonetic scripts represented by a limited set of symbols, for example most European languages such as English. Languages using ideographic scripts such as Chinese or Korean may need a totally different approach \cite{Chinese chord paper}.

The alphabet can be subdivided into two categories – ordered symbols and unordered symbols. Ordered symbols include the alphabet ‘a’, ‘b’, ‘c’, .. and the number ‘0’, ‘1’, ‘2’, ‘3’, ... Most users know that ‘b’ follows ‘a’ in the alphabet and ‘c’ follows ‘b’ etc. This knowledge can be used to improve the interaction. Examples of symbols without a well-established or standard order are the punctuation symbols, basic arithmetic operators etc. In this study only ordered symbols will be considered, but the method extends to unordered symbols as well. The set of symbols A, also known as the alphabet, comprise of the symbols a_1, a_2, .., a_K where K is the size of the alphabet and the index i indicate the rank in the order such that $a_{i-1} < a_i < a_{i+1}$, $\forall a_i \in A$.

2.5. Editing

Advanced editing is included into the model by treating the editing commands as part of the character stream. In order for advanced editing to be incorporated into the model text entry strategy must therefore be equipped with editing symbols, for example BACKSPACE, LEFT, RIGHT, UP, DOWN, INSERT and DELETE to mention a few. Further, case is included by providing TO-UPPERCASE and TO-LOWERCASE symbols or simply a CAPS-LOCK symbol. The particular editor is therefore responsible for interpreting these editing symbols that arrives in the symbol stream.

2.6. Modelling text entry strategies

A static text entry strategy can be modelled using a finite state machine modelled as a directed graph $G(V, E)$, where the vertices V represent text-entry states and the edges E represent transitions between these states. The states typically represent time-intervals when the device is waiting for input from a user while the users are determining their next move. A transition is triggered by a user signal, i.e. when a user presses a key, and the finite state machine moves into a new state. Transitional edges are therefore labeled with a signal label s_i indicating the user signal (or keystroke) that triggers the transition. A transition may also trigger an output signal. This is denoted using the notation $s_i{:}a_j$, where the signal s_i triggers the symbol a_j. For example ‘2:c’ denotes that signal ‘2’ produces the output ‘c’, ‘3:’ denotes that signal ‘3’ leads to a state transition but no symbol is output, ‘:d’ defines a default state transition that occur without a signal but produces the symbol ‘3’ and finally ‘:’ indicates a default state transition without a output symbol.

To simplify the diagram and increase readability end states are denoted by edges that do not point at other states. End states are therefore also easy to identify in a diagram. The graph should always comprise a start state or start vertex. In this paper start states are represented using a gray shaded vertex.

A graph can only have one starting-state but may possess multiple end-states.

Given a test entry strategy defined using graph G, then the shortest path between the two states C_a and C_b is represented using $Path(G, C_a, C_b)$, and the length of this path is $|Path(G, C_a, C_b)|$. We also define an output state $C_{out}(a_i)$ as a state that comprises an output edge representing a transition that produces the output symbol a_i. Finally, the set of exit states C_{exit} are states without outbound edges.

3. Evaluating text-entry strategies

In this section evaluation criteria for mobile text-entry are explored, namely correctness, error recoverability, KSPC and a new measure – mean error recovery distance MERD.

3.1 Correctness

We define the correctness of a text entry strategy to mean that it is both feasible and it has full coverage. A feasible strategy can be implemented within the constraints of the target device, and a strategy has full coverage if all the symbols of in the alphabet can be output.

3.2. Feasibility

For a device to be feasible the following two criteria must be satisfied. First, the number of outbound edges or leaving transitions from a given state must not exceed the number of signals $|S|$ supported by the device. A simple graph traversal can be used to ensure that this constraint is satisfied. Second, each trigger of a state must be unique. If two triggers or more are identical then the design is ambiguous. This constraint can be verified by a simple graph traversal.

3.3. Coverage

For a text entry strategy to cover the entire alphabet there must be at least one unique transition for each symbol of the alphabet. I.e. for each symbol in the alphabet there must be an edge where the symbol is the output.

Further, there must be a path from the start state to all the states where the transitions labelled with the output symbols originate. This can be verified by applying Flynn’s algorithm to compute the distance between any two vertices in the graph. Flynn’s algorithm computes a distance matrix based on an adjacency matrix representation of the graph. Coverage is then ensured if the distance between the start state and all the states that are the origins of output producing transitions are greater than zero. Note that with small alphabets the time complexity of computing the distance matrix is insignificant. Also note that this matrix can be reused for several other evaluation measures discussed in subsequent sections.

3.4. Error recoverability

Although a text entry design has full coverage it might not be possible to fully recover from errors. We define error recovery as the ability to reach any state in the text entry strategy from any other non-exit state. Typically, a user makes a mistake while selecting the desired letter by walking the graph, and then the error recovery characteristics of the graph will allow the user to reach the desired state without restarting from the origin. In terms of graph theory there should be a path from any state to any output state in the graph for the text entry design to have error recovery capabilities. When using Flynn's algorithm (as described in a previous section) all the non-diagonal elements in the distance matrix should be non-zero.

3.5. Mean error recovery distance (MERD)

We propose a measure indicating the average effort required to recover from errors, namely the mean error recovery distance (MERD). Note that MERD is only a simple distance oriented measure and does not incorporate any cognitive factors. Although cognitive factors are not included the MERD can serve as a best-case and identify poor text entry strategies.

The states on the path from the start-state to the output state associated with the desired symbol a is given by $P = Path(G, C_{start}, C_{out}(a))$. Imagine that the user has made a mistake and ended up in a state not on this path. We define the set of states not on the path from C_{start} to $C_{out}(a)$ that are not exit states as $E = P'/C_{exit}$, i.e. the compliment of the set of states on the path that are not exit states. The MERD associated with reaching the output state associated with symbol a when being in a state not on the path of a is:

$$MERD(a) = \frac{1}{|E|}\sum_{c \in E} |Path(G, c, C_{out}(a))| \qquad (1)$$

In other words, the average distance from the states not on the path from the start state to the output state of the symbol and to the output state of the symbol is computed. The overall MERD is computed as:

$$\overline{MERD} = \sum_{\forall a \in A} MERD(a)\, f(a) \qquad (2)$$

Namely, the sum of the average recovery distance for each symbol of the alphabet multiplied by their respective probability of occurrence.

3.6. KSPC

KSPC (keystrokes per character) is a measure proposed by MacKenzie [15-17], which indicates the number of keystrokes required in order to retrieve a particular character. The KSPC measure usually refers to the average KSPC, the minimum KSPC and maximum KSPC. Obviously, KSPC indicates the potential speed in which text can be typed. However, KSPC is criticised in the human computer interaction community for being over simplistic not capturing other important factors affecting typing speed. We do not wish to add to this debate but rather demonstrate how to compute KSPC using the proposed methodology.

Given a graph model of the text interface it is easy to determine minimum maximum and average KSPC. Given a starting state C_{start} and a set of output states $C_{out} \subseteq C$, then the minimum KSPC is given by:

$$KSPC_{\min} = \min_{a \in A} |Path(G, C_{start}, C_{out}(a))| \qquad (3)$$

The maximum KSPC is given by:

$$KSPC_{\max} = \max_{a \in A} |Path(G, C_{start}, C_{out}(a))| \qquad (4)$$

and finally the average KSPC is given by:

$$\overline{KSPC} = \sum_{a \in A} |Path(G, C_{start}, C_{out}(a))|\, f(a) \qquad (5)$$

where $f(a)$ is the probability of occurrence for the character a.

4. Examples of the methodology applied

For the purpose of demonstrating the proposed methodology six text entry strategies are modelled, namely the DateStamp approach (see Figure 1), a new one-way DateStap strategy (see Figure 2), Raghunath and Narayanaswami's [6] wristwatch strategy (see Figure 3), multi-tap (see Figure 4), Sandnes et al's [7] multi-ring (see Figure 5) and a mesh (see Figure 6). The characteristics of each strategy is summarised in Table 1.

The graph model for the date-stamp method for an alphabet of five characters ('a', 'b', 'c', 'd' and 'e') is presented in Figure 1. It is obvious how this extends to the full alphabet (omitted to save space). The user start in the state indicated with the gray background, and the user can move to the two neighboring states using key 1 or 2. To select the character associated with the state the user presses key 3. Clearly, this design is feasible as each state only has three outbound edges. Further, the design has full coverage as there is a path from the start state to all the other states. In fact, there is a path from any non-exit state to any other state and the design supports full error correction with a *MERD* = 5.01.

Further, $KSPC_{min}$ = 1 as only one transition is needed to produce the output character 'a' (assuming snap-to-beginning). $KSPC_{max}$ of this design is identical to the diameter of the graph [18]. The diameter of a graph is defined as the longest path between any two vertices of a graph. For this particular design $KSPC_{max}$ = 3. For the English alphabet $KSPC_{max}$ = 14. The mean number of keystrokes per character for the English alphabet is *KSPC* = 7.77.

The graph representation simplifies the understanding of text entry designs, and in this next example shown in Figure 2 a subtle but significant alteration is made to the previous date-stamp approach (This is to the best of our knowledge a yet undocumented strategy). The difference between this strategy and the date stamp approach is that the user only can scroll in one direction, and that the user at each step can select one of two characters. The example design in Figure 3 also comprises 5 states, but it supports twice as many output symbols (a total of 10 symbols). The design is feasible and has full coverage. Clearly, the $KSPC_{min}$ = 1, and $KSPC_{max}$ = 5 for this design and $KSPC_{max}$ = 14 for the full English alphabet and the mean *KSPC* = 6.113. This strategy is thus theoretically more efficient than the traditional date-stamp approach. The strategy is error recoverable. However, the drawback is that the user can only scroll in one direction and the user may have to traverse the entire ring in order to make a correction. Although the maximum distance needed to recover from error is 12 for both the one-way datestamp and the datestamp strategy, the mean error recovery distance is larger for the one-way datestamp strategy (*MERD* = 6.5) than it is for the datestamp method (*MERD* = 5.01). However, if one for instance assumes a 5% error rate then the benefits of the decrease in KSPC for the one-way datestamp outweighs the drawbacks associated with its increased MERD.

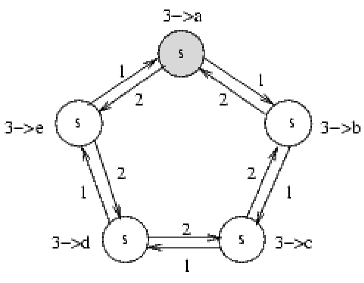
Figure 1: The classic date-stamp text entry strategy

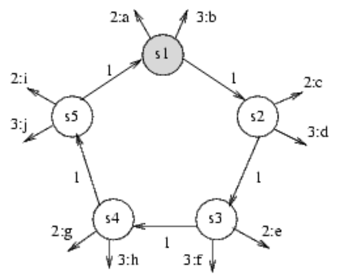
Figure 2: The one-way date-stamp text entry strategy

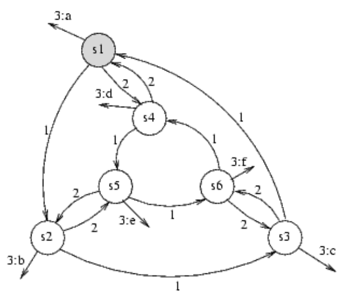
Figure 3: The Raghunath and Narayanaswami wristwatch text entry strategy

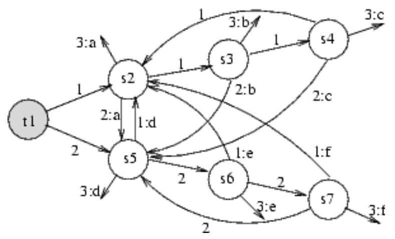
Figure 4: The multi-tap text entry strategy

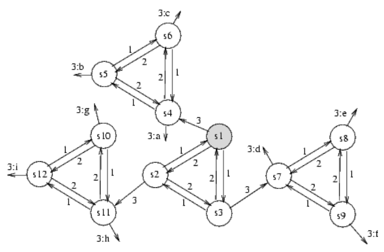
Figure 5: The multi-ring text entry strategy

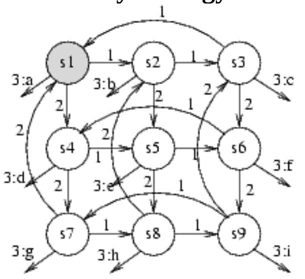
Figure 6: The mesh text entry strategy

The multi-tap approach is a strategy that is widely used on mobile phones. The design in Figure 4 incorporates the characters 'a' to 'f' where three letters is assigned to each of the two of the keys and the third key is used as a break key. To expand this design to the full alphabet more letters are simply added to each of the multi-tap keys. The design is feasible and has full coverage. However, this strategy is not fully error recoverable. For example if the user starts by pressing key 1 the user is moved to state 2. Imagine that this is a mistake as the user intends to type the character 'd'. There is no way go get to state 5 without producing output. If for instance the user chooses to press key 2 to reach state 5 the character 'a' is output. Hence, MERD is not applicable.

Although the multi-ring strategy shown in Figure 5 is correct it is not error recoverable. Once the user enters one of the sub-rings there is no path back. For example, imagine that the user want to enter character 'd'. The user by accident presses key 3 from the start state 1 which moves the user to state 4. State 4 gives the user access to one of the characters 'a', 'b' or 'c'. There is no path from state 4 to state 7, which is used to produce the letter 'd'. Consequently, MERD is not applicable.

Table 1: Summary of the characteristics for the six text entry strategies

Test entry strategy	min KSPC	max KSPC (fig)	max KSPC	mean KSPC	MERD
Datestamp	1	3	14	7.77	5.01
Uni d.s.	1	5	14	6.11	6.50
Rag. watch	1	3	16	7.19	6.20
Multitap	1(2)	4	14	6.69	N/A
Multi-Ring	2	4	7	4.98	N/A
Mesh	1	5	9	4.50	4.50

The mesh strategy shown in Figure 6 is not previously documented. It is a simplification of the bidirectional wraparound mesh [12] for five-key devices. Three characteristics of the mesh are particularly suitable for mobile text entry: each vertex has few outbound edges (i.e. buttons), the diameter of the mesh structure is relatively small and there is a path between any two states in the structure (error recoverability with a short distance). Figure 7 shows a 3x3 mesh incorporating the characters ‘a’ to ‘i’. In this design one key is used to cycle vertically, the second to scroll horizontally and the third key is used to select letters. Obviously, the strategy is feasible and has full coverage. Table 1 shows that this strategy is the best when considering the mean *KSPC* and *MERD*, which are both 4.5 (note that these values depends slightly on the character layout). Only the MultiRing has lower $KSPC_{max} = 7$, as opposed to $KSPC_{max} = 9$ for the mesh.

5. Summary

A graph based methodology for the representation, design and evaluation of text entry techniques for miniature mobile devices is presented. The technique allows different text entry strategies to be compared. Further, it is easy to check for correctness and error recoverability KSPC and a new error recoverability measure ERP is proposed. The methodology is not intended as a replacement for typing test using real people on real devices but rather as an early screening tool as it will identify poor text entry strategies early before the interaction engineer commits to expensive testing.s

References

[1] P. M. Fitts, "The information capacity of the human motor system in controlling amplitude of movement," *Journal of Experimental Psychology*, vol. 47, pp. 381-391, 1954.

[2] S. H. Levine, "Multi-character key text entry using computer disambiguation," in the proceedings of RESNA 10th annual conference, San Jose, California, 1987.

[3] J. G. Kreifeldt, "Reduced keyboard designs using disambiguation," in the proceedings of The human factors society 33rd annual meeting, 1989.

[4] M. T. King, "JustType. TM. - Efficient communication with eight keys," in the proceedings of Proceedings of the RESNA 95¨Annual conference, Vancouver, BC, Canada, 1995.

[5] I. S. MacKenzie, "Mobile Text entry using three keys," in the proceedings of NordCHI'02, pp. 27-34, 2002.

[6] M. T. Raghunath and C. Narayanaswami, "User Interfaces for Applications on a Wrist Watch," *Persona7-30l and Ubiquitous Computing*, vol. 6, pp. 17-30, 2002.

[7] F. E. Sandnes, H. W. Thorkildssen, A. Arvei, and J. O. Buverud, "Techniques for fast and easy mobile text-entry with three-keys (non-dictionary based)," in the proceedings of HCISS'37 Hawaiian International Conference on System Science, Big Island, Hawaii, 2004.

[8] T. Evreinova, G. Evreino, and R. Raisamo, "Four-Key Text Entry for Physically Challenged People," 2004.

[9] K. Tamaka-Ishii, Y. Inutsuka, and M. Takeichi, "Entering ext with A Four-Button Device," in the proceedings of The 19th International Conference on Computational Linguistics COLING 2002, Taipei, Taiwan, pp. 988-994, 2002.

[10] J. Moyes, "Chord Keyboards," *Applied Ergonomics*, vol. 14, pp. 55-69, 1983.

[11] P. Isokoski and R. Raisamo, "Device independent text input: A rationale and an example," in the proceedings of the Working Conference on Advanced Visual Interfaces AVI2000, Palermo, Italy, 2000.

[12] T. Bellman and I. S. MacKenzie, "A probabilistic character layout strategy for mobile text entry," in the proceedings of Graphics Interface '98, Toronto, Canada, pp. 168-176, 1998.

[13] J. J. Darragh, I. H. Witten, and J. M. L., "The reactive keyboard: a predictive typing aid," *IEEE Computer*, vol. 23, pp. 41-49, 1990.

[14] J. Lehikoinen and M. Roykee, "Lehikoinen, J. and Roykee, M.",," *Interacting withComputers*, vol. 13, pp. 601-625, 2001.

[15] I. S. MacKenzie, "KSPC (keystrokes per character) as a characteristic of text entry techniques," in the proceedings of Mobile HCI 2002, 2002.

[16] M. Silfverberg, I. S. MacKenzie, and P. Korhonen, "Predicting text entry speed on mobile phones," *ACM CHI'2000*, vol. 1, pp. 9-16, 2000.

[17] C. L. James and R. K. M., "Text Input for mobile devices: Comparing Model prediction to actual performance," in the proceedings of CHI'2001, pp. 365-371, 2001.

[18] V. Kumar, A. Grama, A. Gupta, and G. Karpypis, *Introduction to Parallel Computing - Design and analysis of algorithms*: The Bennjamin/Cummings Publishing Company, Inc, 1984.