MAINTAINING PREFERENCE NETWORKS THAT ADAPT TO CHANGING PREFERENCES

by

Ki Hyang Lee

Bachelor of Computer Science, University of New Brunswick

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
the Requirements for the Degree of

Master of Computer Science

in the Graduate Academic Unit of Computer Science

Supervisor: Michael Fleming, PhD, Computer Science
Scott Buffett, PhD, Computer Science

Examining Board: Huajie Zhang, PhD, Computer Science, Chair
Weichang Du, PhD, Computer Science
Donglei Du, PhD, Business Administration

This thesis is accepted by the Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

September 2012

©Ki Hyang Lee, 2012

DEDICATION

To my parents

Youngil Lee and Hansoon Kim

ABSTRACT

In recent years, online transaction systems have used automated decision making or automated negotiation technology to interact with their customers by suggesting other products or services that might have high utility for those customers. However, decision making can be more difficult with an enormous amount of information, not only for the human but also for the automated decision making processes. Without a proper user preference model, search results are often inaccurate, and thus may result in many pages of output for users to examine. Although most user preference elicitation models have been developed based on the assumption that user preferences are stable, user preferences may change in the long term and may evolve with experience, resulting in dynamic preferences. Therefore, in this thesis, we provide a model called a Dynamic COP-net that is maintained using an approach that does not require the entire preference graph to be rebuilt when a previously-learned preference is changed, with efficient algorithms to add new preferences and to delete existing preferences. A Dynamic COP-net can maintain the preference model without rebuilding the entire preference model whenever there is a new preference added or deleted, within $O(n^2)$ time. Dynamic COP-nets are shown to outperform existing algorithms for insertion, especially for large numbers of attributes and for dense graphs. They do have some shortcomings in the case of deletion, but only when there is a small number of attributes or when the graph is particularly dense.

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Dr. Scott Buffett and Dr. Michael Fleming, who were the biggest contributors during my graduate studies. They both gave me a vision and helped me to develop my own creative ideas. Their support cannot be countable. I cannot imagine my master's work being done without their remarkable and kind support. Both Dr. Scott Buffett and Dr. Michael Fleming always like to share their great ideas with me even though I had been novice to expressing research work. They have always been the best supporters during my graduate studies. I will never forget their efforts to make my master's work done. Once again, special thanks to the visions and enthusiasm that my supervisors showed me.

Producing a master's thesis really needs sacrifices which are rarely made alone. I would like to thank my husband, Sangwhan Cha, for all the sacrifices he has been forced to make for my research work. Thanks to my lovely kids, Emily and Daniel Cha, who have always given me a big smile so that I can feel so comfortable and happy anytime. Finally, thanks to my father in Korea who always loves me deeply.

Table of Contents

DEDICATION ii

ABSTRACT iii

ACKNOWLEDGEMENTS iv

Table of Contents v

List of Tables vii

List of Figures viii

1. Introduction 1

2. Background Information 4

2.1 Preferences and Utility Theory 4

2.1.1 Preference relations 5

2.1.2 Utility Functions 6

2.2 Evolving Preference Models 8

2.2.1 Evolving preferences 8

2.2.2 Evolving Preference Models 9

2.3 Preference Graphs 11

2.3.1 CP-nets 11

2.3.2 Conditional Outcome Preference Networks 14

2.4 Transitive Reduction Maintenance 21

2.4.1 Graphs 21

2.4.2 Transitive reduction vs. transitive closure 22

2.4.3 Facilitating Transitive Reduction Maintenance 23

3. Dynamic COP-nets 30

3.1 Representation of preferences with COP-nets 31

3.2 Algorithm for Maintaining Transitive Reduction 32

3.2.1 New preference insertion 35

3.2.2 Preference deletion 41

4. Evaluation and Analysis 49

4.1 Evaluation 49

4.1.1 Evaluation Plan 49

4.1.2 Reduced COP-networks vs. Unreduced COP-networks 51

4.1.3 Proposed vs. Current methods 52

4.1.3.1 Initial Construction time 52

4.1.3.2 Average update time 56

4.1.3.3 Construction success rate within a given time limit 63

4.1.3.4 Average number of updates with large number of attributes in a given time 64

4.2 Analysis 69

5. Conclusions and Future work 70

5.1 Summary 70

5.2 Thesis Contributions 72

5.3 Future Work 73

Bibliography 74

Curriculum Vitae

List of Tables

Table 2.1: Node representation for outcomes 16

Table 2.2: Node representation for outcomes 17

Table 4.1 Average time of finding longest path (ms) 51

Table 4.2 Initial construction time with different densities 53

Table 4.3 Initial construction time with different numbers of attributes 54

Table 4.4 Initial construction time for sparse graphs 55

Table 4.5 Initial construction time for dense graphs 55

Table 4.6 Average insertion time with different densities (8 attributes) 57

Table 4.7 Average insertion time with different numbers of attributes 57

Table 4.8 Average insertion time with sparse graphs 58

Table 4.9 Average insertion time with dense graphs 59

Table 4.10 Average deletion time with different densities (8 attributes) 60

Table 4.11 Average deletion time with different numbers of attributes 61

Table 4.12 Average deletion time with sparse graphs 61

Table 4.13 Average deletion time with dense graphs 62

Table 4.14 Construction success rate with different # of attributes in 500ms 63

Table 4.15 Average number of insertions with different densities (9 attributes) in 500ms 64

Table 4.16 Average number of insertions in sparse graphs in 500ms 65

Table 4.17 Average number of deletions with different densities (9 attributes) in 500ms66

Table 4.18 Average number of deletions in sparse graphs in 500ms 67

List of Figures

Figure 2.1: (a) CP-net for "My Dinner I"; (b) the induced preference graph [8] 12

Figure 2.2: (a) CP-net for "My Dinner II"; (b) the induced preference graph [8] 13

Figure 2.3: (a) the induced preferences for "My Dinner I"; (b) the initial COP-net 16

Figure 2.4: (a) the initial COP-net; (b) the reduced COP-net 17

Figure 2.5: (a) the induced preferences; (b) the initial COP-net 18

Figure 2.6: (a) the initial COP-net; (b) the reduced COP-net 19

Figure 2.7: COP-net showing elicited and estimated utilities 20

Figure 2.8: Reachability 23

Figure 2.9: Input matrix example 24

Figure 2.10: Reachable from k to m with new edge (i, j) insertion 24

Figure 2.11: Inserting a single edge 27

Figure 2.12: King's insert algorithm 28

Figure 2.13: King's delete algorithm 28

Figure 3-1 Four basic conditions between two vertices 32

Figure 3-2 Dynamic COP-net example 1 34

Figure 3-3 Dynamic COP-network with reduced COP-net 35

Figure 3-4 Proposition 3.1, Proposition 3.2 and Proposition 3.3 36

Figure 3-5 Insertion of a new arc (v_i, v_j) 36

Figure 3-6 Insertion algorithm for dynamic COP-nets 37

Figure 3-7 Insertion algorithm: Step 1 38

Figure 3-8 Dynamic COP-network: Example 2 38

Figure 3-9 Insertion algorithm (Example 2): Step 1 39

Figure 3-10 Insertion algorithm (Example 2): Step 2 40

Figure 3-11 Insertion algorithm (Example 2): Step 3 40

Figure 3-12 Dynamic COP-network with Reduced COP-net 41

Figure 3-13 Proposition 3.4 42

Figure 3-14 Proposition 3.5 43

Figure 3-15 Graphical Sets Information for a given $i_1 \in I_1$ 44

Figure 3-16 Deletion algorithm for Dynamic COP-nets 45

Figure 3-17 Deletion algorithm (Example 2): Step 1 46

Figure 3-18 Deletion algorithm (Example 2): Step 2 47

Figure 3-19 Deletion algorithm (Example 2): Step 3 48

Figure 3-20 Dynamic COP-network with Reduced COP-net 48

Figure 4.1 Finding longest path in reduced and unreduced networks 52

Figure 4.2 Initial construction time with different densities 54

Figure 4.3 Initial construction time with different numbers of attributes 54

Figure 4.4 Initial construction time for sparse graphs 55

Figure 4.5 Initial construction time for dense graphs 56

Figure 4.6 Average insertion time with different density graphs 57

Figure 4.7 Average insertion time with different numbers of attributes 58

Figure 4.8 Average insertion time with sparse graphs 58

Figure 4.9 Average insertion time with dense graphs 59

Figure 4.10 Average deletion time with different density graphs 60

Figure 4.11 Average deletion time with different numbers of attributes 61

Figure 4.12 Average deletion time with sparse graphs 62

Figure 4.13 Average deletion time with dense graphs 62

Figure 4.14 Construction success rate with different # of attributes in 500ms..... 63

Figure 4.15 Average number of insertions with different density graphs (9 attributes)... 64

Figure 4.16 Average number of insertions in sparse graphs in 500ms 65

Figure 4.17 Ratio of insertions in sparse graphs in 500ms 66

Figure 4.18 Average number of deletions in different density graphs (9 attributes)..... 67

Figure 4.19 Average number of deletions with sparse graphs in 500ms 67

Figure 4.20 Ratio of deletions in sparse graphs in 500ms 68

1. Introduction

Nowadays, more business transactions occur through the internet and more people interact with online transaction systems to purchase products. Those online transaction systems may use automated decision making or automated negotiation technology to interact with their customers by suggesting other products or services that might have high utility for those customers. However, decision making can be incredibly difficult when faced with an enormous amount of information, not only for the human but also for the automated decision making processes. For example, in amazon.com, there are millions of items for sale at a time, and it seems almost impossible for a buyer to navigate all the items by hand to find the books that he/she wants to buy. Even when product search tools are applied, without a proper user preference model, search results are often inaccurate, and thus may result in many pages of output for users to examine. Therefore, in this thesis, we propose a dynamic preference model that starts with a small amount of information about a user's preferences and that can be modified as those preferences change.

Preference elicitation has been studied to help automated decision making or automated negotiation by extracting knowledge of a user's preferences in order to maximize utility over the possible outcomes [1]. Many approaches related to preference elicitation or utility elicitation have been proposed to help with making decisions more efficiently. Various methods for reasoning with these preferences have been studied to help to easily solve preference-related problems, such as determining whether a user prefers one outcome over another or estimating utilities for the entire set of outcomes.

The Conditional Preference Network (CP-net), proposed by Boutilier et al. [8], is a well known graphical representation of conditional preference relations. Conditional preferences are those where tastes are dependent on the outcome of one or more particular attributes. For example, suppose there is a car with two attributes "Body Type" and "Color". A user has preferences for "Body Type" such as "SUV is preferred over Sedan" and for "Color" such as "Black is preferred over Silver if Body Type is Sedan, and Silver is preferred over Black if Body Type is SUV". In this case, the preference for "Color" is dependent upon the outcome for "Body Type", making this a *conditional preference*. CP-nets are a relatively compact, intuitive, and structured specification of preference relations. While CP-nets are used to reason about conditional preferences over values within a subset of attributes, a Conditional Outcome Preference Network (COP-net), proposed by Chen et al. [2], has extended conditional preference representation for representing user preferences and for predicting preferences over all feasible outcomes for any type of conditional preference.

These approaches are based on the assumption that a user's preferences are stable. However, it has been recognized that preferences change, especially in the long term, affected by human genetic evolution, technological change, the evolution of social systems, and the changing availability of environmental and other natural resources [5]. Preferences may evolve with experience, resulting in flexible future preferences [6]. Also, a user's preferences are likely to change as a result of the elicitation process itself [13]. Without dynamic preference models capable of evolving over time, if there is a new preference added or changed, we may have to rebuild the entire preference model.

The goal of this thesis is to extend the current state of preference modeling to include flexible future preferences and to represent changing preferences. In this thesis, we provide for situations in which users might have changing preferences. Dynamic COP-nets are proposed as an approach for maintaining preference models that does not require the entire preference model to be rebuilt when a previously-learned preference is changed. We compared our proposed model to current existing approaches and demonstrated its effectiveness through experimentation.

In Chapter 2, we introduce some background information that is used in this research such as changing preferences, transitive reduction, preference modeling and utility elicitation. In Chapter 3, we discuss a new methodology for maintaining Dynamic COP-nets, introducing new insertion and deletion algorithms. In Chapter 4, we discuss test results of the new methodology and how it compares to the current methodology. Finally, conclusions and possible future work are presented in Chapter 5.

2. Background Information

In this chapter, we will explain the concepts and other current approaches related to our research such as preferences and utility theory, evolving preference models, preference graphs and transitive reduction maintenance.

2.1 Preferences and Utility Theory

Preferences have played an important role in automated decision making. In this thesis, we use *ceteris paribus* preferences, meaning "everything else being equal," whose statements concisely represent preferences over outcomes in a natural way to human thinking [8]. This idea is expressed very well in the opening of Hansson's paper:

When discussing with my wife what table to buy for our living-room, I said: "A round table is better than a square one." By this I did not mean that irrespectively of their other properties, any round table is better than any square-shaped table. Rather, I meant that any round table is better (for our living-room) than any square table that does not differ significantly in its other characteristics, such as height, sort of wood, finishing, price, etc. This is preference *ceteris paribus* or "everything else being equal". Most of the preferences that we express or act upon seem to be of this type. [18]

Based on the way humans reason about preferences, there are qualitative representations and quantitative representations. A qualitative representation starts with reasoning about the relative desirability of each outcome or subset of outcomes, whereas a quantitative representation typically starts with numeric valuations of each attribute, direct estimation of a utility function or detailed estimation of probabilities [17]. In many domains, a

qualitative approach is more natural and appropriate to start with than a quantitative approach. Doyle et al. in 1991 introduced a qualitative preference elicitation process, a theoretical formulation of human preferences, to generate preference statements in terms of *ceteris paribus* preferences [20]. McGeachie et al. presented methods for translating preference information from a qualitative representation of Doyle's theoretical formulation of human preferences under a *ceteris paribus* assumption to a quantitative representation which gained an ordinal utility function [17]. In Section 2.1.1, we will define the notions of preference relations in terms of a qualitative preference representation. In Section 2.1.2, we will discuss utility theory in detail.

2.1.1 Preference relations

In this section, we will discuss an outline of relevant notions related to preference relations. All notations in this section refer directly to Boutilier et al.'s paper in 2004 [8]. Boutilier starts with assuming the world can be in one of a number of states $\boldsymbol{S}$ and at each state $\boldsymbol{s}$ there are a number of actions $\boldsymbol{T}$ that can be performed. Each action has a specific outcome and the set of all outcomes is denoted by $\boldsymbol{O}$. There are 3 possible preference relationships between outcomes o_1 and o_2:

$o_1 \succeq o_2$ means that, for the decision maker, o_1 is equally or more preferred when compared to o_2.

$o_1 \succ o_2$ means that the decision maker strictly prefers o_1 over o_2.

$o_1 \sim o_2$ means that the decision maker is indifferent between o_1 and o_2.

A set of attributes $\boldsymbol{A} = \{X_1, X_2, \ldots, X_n\}$ is defined over which the decision maker has preferences. Each attribute X_i is associated with a domain $Dom(X_i) = \{x_1^i, x_2^i, \ldots, x^i_{n_i}\}$ of

values it can take. An *assignment* **x** of values to a set $\mathbf{X} \subseteq \mathbf{A}$ of attributes is a function that maps each attribute in **X** to an element (value) of its domain. *Asst*(**X**) refers to the set of all assignments to $\mathbf{X} \subseteq \mathbf{A}$. If **X**=**A** then **x** is a *complete assignment*; otherwise **x** is a *partial assignment*. Then, a complete assignment represents an outcome. Given a problem over n attributes with domains $Dom(X_1),\ldots, Dom(X_n)$, there are $|Dom(X_1)| \times \ldots \times |Dom(X_n)|$ assignments, and thus direct assignment of a preference function is usually infeasible due to its exponential number of outcomes. Therefore it is important to consider the idea of *preferential independence*.

A set of attributes **X** is *preferentially independent* of its complement **Y** = **A-X** if and only if the preference relations over assignments to **X**, when all other attributes are held fixed, are the same no matter what values the other attributes from **Y** take. For all $x_1, x_2 \in$ *Asst*(**X**) and $y_1, y_2 \in$ *Asst*(**Y**), we have $x_1y_1 \succeq x_2y_1$ iff $x_1y_2 \succeq x_2y_2$. Furthermore, we can say x_1 is preferred to x_2 *ceteris paribus*. It provides computational benefits, since preferentially independent sets can be considered without regard to the other preference sets.

X is *conditionally preferentially independent* of **Y** given an assignment **z** to **Z** if and only if **X** is preferentially independent of **Y** when **Z** is assigned **z**. For all $x_1, x_2 \in$ Asst(**X**) and $y_1, y_2 \in$ Asst(**Y**), we have $x_1y_1z \succeq x_2y_1z$ iff $x_1y_2z \succeq x_2y_2z$.

2.1.2 Utility Functions

Utility represents a degree of preference in terms of numeric valuations. Utility functions have been applied to quantitative representation of human preferences, which is useful when the primary attributes of preferences are quantitative measures, or when outcomes

can be defined with easily measurable quantities [17]. Models considering quantitative representations of preference relations make the preference elicitation process easier by imposing specific requirements on the form of the utility function [8]. For example, some products whose preferences highly depend on the prices might be best organized by sorting the prices at the start and then working with these easily sorted "keys" throughout the rest of the process, rather than repeatedly re-determining pairwise comparisons. When the argument of the utility function is a vector indicating values for several attributes, it is called a multi-attribute utility function, in which the vector indicates a value for each attribute, which is useful in addressing the issues of uncertainty and tradeoffs between conflicting objectives [19]. The outcomes with higher utility indicate the more preferable outcomes. Once a user preference model has been represented with a graph structure, utility in a graph can be relatively represented with the distance between two nodes. Often a utility function assigns a real number, between 0 and 1, to each outcome. A utility of 1 is assigned to the most preferred outcome, often represented by the topmost node in a graph, and a utility of 0 indicates the least preferred outcome, often represented by the bottommost node in a graph. We use $u(o_i)$ to represent the utility of an outcome o_i. According to the principles of utility theory [20], if a user's preferences follow certain utility rules, then there exists a real valued utility function that operates on outcomes such that $u(o_i)>u(o_j)$ if and only if o_i is preferred to o_j, $u(o_i)=u(o_j)$ if and only if the user is indifferent between o_i and o_j, and $u(o_i)\geq u(o_j)$ if and only if o_i is preferred over o_j or the user is indifferent between the two outcomes.

2.2 Evolving Preference Models

In this section, we will discuss evolving preferences and current research on evolving preference models. Although most research on modeling human preferences has been developed on a basic assumption of preference stability, the idea of more flexible and dynamic preference models has been considered [6, 10]. First in Section 2.2.1, we will review existing research on how human preferences evolve. In Section 2.2.2, we will look over some of the current approaches for modeling evolving or changing preferences.

2.2.1 Evolving preferences

Preferences have often been considered to be stable. Many economists assume that an individual facing the same choice at a different time would express the same preference [5], and thus preference elicitation models have been developed based on this assumption. However, the possibility that preferences change has been considered.

According to Koopmans, preferences may evolve with experience, resulting in flexible future preferences [6]. Some economists have recognized that preferences are influenced by endogenous factors such as others' consumption patterns or past experiences [5], and thus if human behaviour depends on past experiences and if such behaviour is governed by adaptive economizing procedures, the assumptions of the existence of long-run static utility functions may be challenged.

A user's preferences are likely to change as a result of the elicitation process itself, such as in the following situations. Users are not aware of all preferences until they validate

them all. For example, a user might not care about intermediate airports until a solution includes any place that he/she dislikes. Also, preferences are often in contradiction with each other and require the user to make tradeoffs. From the previous example, a user may add a new preference after checking intermediate airports and the new preference may cause a conflict with the previous preference. In tradeoffs, users may add, remove or change preferences at any time during the preference elicitation process. This cannot be supported by a preference model that requires preferences to be predefined by a user initially and expects these preferences to be stable [13].

Besides these dynamic properties of a user's preferences, the decision making environment itself may also change with time. For example, in a travel agent system, the tickets for some specific flights may be sold out during a user's decision making process, and thus the user has to modify his or her preferences if the results from the previous preferences do not satisfy the user anymore [13]. Therefore, if a preference representation mechanism does not consider changing preferences, it is required to rebuild the entire preference model whenever there is any change in the initial preference model.

2.2.2 Evolving Preference Models

While Boutilier's Conditional Preference networks (CP-nets, for short) [7, 8] as well as Chen's Conditional Outcome Preference networks (COP-nets, for short) [2] have been presented based on the assumption that a user's preferences are stable, Michael McGeachie [10], in 2005, presented a method for recompiling utility functions with *ceteris paribus* preferences in a dynamic domain. McGeachie [17], in 2004, suggested a

framework, a *model graph,* a directed graph which represents preferences that takes a set of ceteris paribus preferences (*other things being equal*) and compiles a utility function consistent with those preferences. The concept of a model graph is similar to Boutilier's CP-nets with extension, in that a model graph can represent the preference with cross attributes (more than one feature at a time). Therefore when Boutilier's CP-nets cannot express a statement such as $p \succ q$, a model graph can express this statement. In a model graph, the feature space represents a domain of the preference model. Later in 2005, MaGeachie extended a model graph by suggesting recompiling utility functions to support a dynamic domain.

Dynamic domains embody changes in preferences. Those changes include the following. First, the preferences themselves may change. Second, the variables over which preferences are expressed may change. Third, the forms of utility independence that hold in the domain may change. With the classical decision theory that is based on preference stability assumptions, preference models can be too limited or too rigorous to support decision-making in dynamic domains and under changing conditions. The properties that are amenable to functioning in a dynamic domain are the following. First, it allows some flexibility in the way a user is able to specify preferences, by allowing incomplete preferences, partial attributes in the domain, and conditional and/or unconditional utility dependencies and utility independencies. Also, it allows adding or removing attributes to and from the domain without altering or invalidating the existing preferences.

The recompiling utility function is based on the utility independence assumption. A set $\boldsymbol{S}$ of *ceteris paribus* preferences is considered, as well as which utility independence assumptions are consistent with $\boldsymbol{S}$. Then, a utility function is considered as a linear combination of subutility functions, with each of these computed from model graphs of subsets of the feature space with a set of independent preferences. This method allows two types of preference changes: utility independence changes and changes in the underlying feature space, which is the dynamic domain.

2.3 Preference Graphs

In this thesis, we refer to the most well known graphical preference model called CP-nets and its extended model COP-nets. In Section 2.3.1, we will discuss CP-nets in more detail and in Section 2.3.2, we will discuss COP-nets and how they compare to CP-nets.

2.3.1 CP-nets

Boutilier et al., in 1999, presented a structure, CP-nets, for reasoning about conditional preferences reflecting conditional dependence and independence of preference statements under the *ceteris paribus* (all else being equal) assumption [7, 8]. The representation is based on dependent and conditional preferences, independent from other attributes. Each node represents an attribute in a domain. For each attribute, the user must identify a set of parent attributes (if any) whose values can influence the user's preference over values for the attribute. Therefore, each directed edge (u, v) in a CP-net represents that preferences over attribute v are dependent on values for attribute u. For each attribute $\boldsymbol{A}$, a *conditional*

preference table (CPT) is used to describe the user's preferences over the values of the attribute A given every combination of parent values.

Figure 2.1 shows the simple example of a CP-net from Boutilier et al.'s paper in 2004 [8]. This CP-net consists of two attributes S and W, standing for soup and wine. The user, in this example, strictly prefers fish soup (S_f) to vegetable soup (S_v) and has conditional preferences for red (W_r) and white(W_w) wine that depend on the soup: The user prefers red wine if a vegetable soup is served, and white wine if a fish soup is served. These two conditional preferences are represented using the notation S_v: $W_r \succ W_w$ and S_f: $W_w \succ W_r$. Figure 2.1(a) shows its preference relations with CPTs and Figure 2.1(b) shows the preference graph over outcomes induced by this CP-net. In this example, the fact that $S_v \wedge W_r$ is preferred to $S_v \wedge W_w$ is indicated by the direct arc from $S_v \wedge W_w$ to $S_v \wedge W_r$. The top element ($S_v \wedge W_w$) is the worst outcome while the bottom element ($S_f \wedge W_w$) is the best.

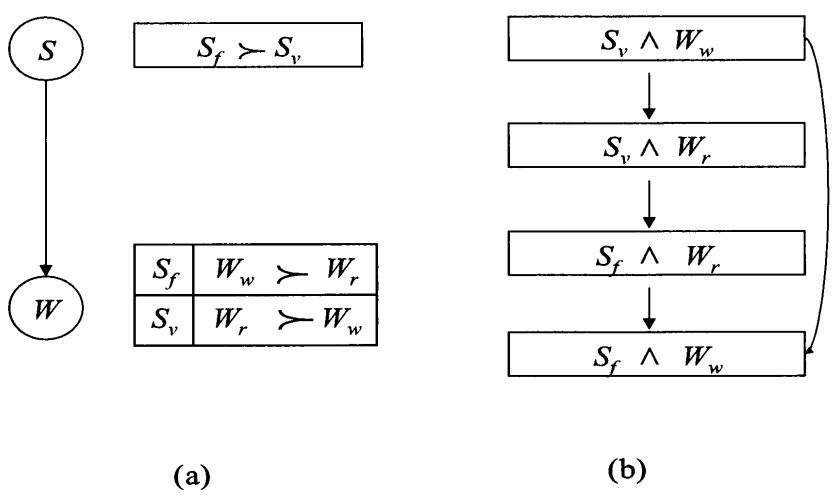


Figure 2.1: (a) CP-net for "My Dinner I"; (b) the induced preference graph [8]

Figure 2.2 [8] shows another example, which extends the chain CP-net of "My Dinner I" in Figure 2.1 from Boutilier et al.'s paper in 2004. In this example, the user adds the main course ***M*** as another attribute and adds a new preference: I strictly prefer a meat course M_{mc} to a fish course M_{fc}. In addition, the user prefers not to have two fish courses in one dinner, and thus the preference between vegetable and fish soup is conditioned on the main course. The user prefers fish soup if the main course is meat and prefers vegetable soup if the main course is fish. In this example, when a new attribute is strictly added to the previous ones, the previous preference relations are conditioned on the new attribute.

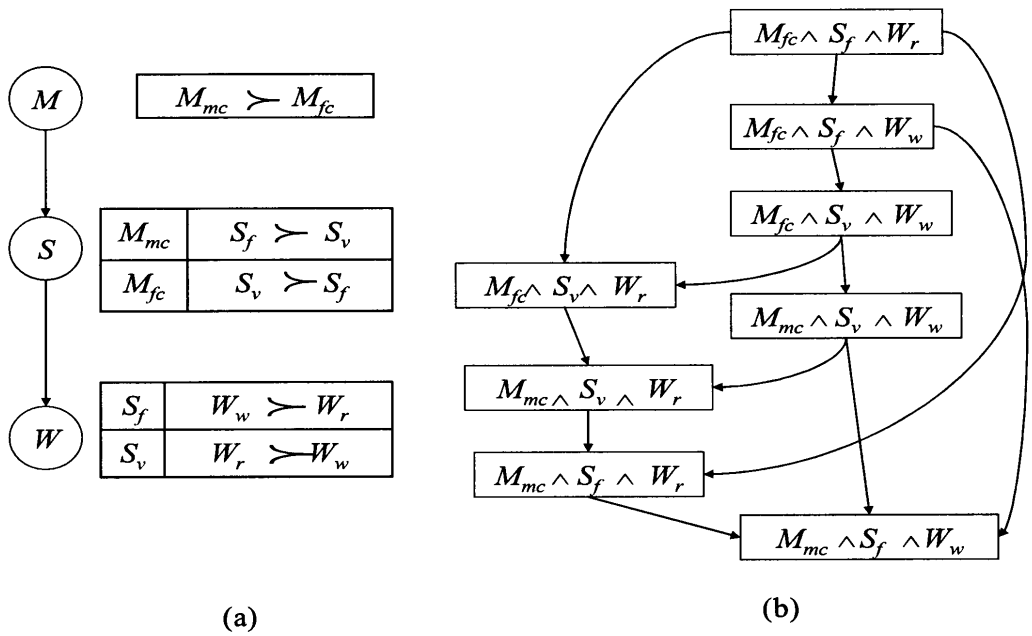


Figure 2.2: (a) CP-net for "My Dinner II"; (b) the induced preference graph [8]

Boutilier et al. consider uncertainty with preference relations by offering an appropriate tradeoff between flexible preference expression and a particular preference structure based on a basic assumption of preference consistency. CP-nets mainly focus on flexible

preference expression and a preference structure that considers conditional preference statements without any suggestion on the conflict issue on the preference relation model.

2.3.2 Conditional Outcome Preference Networks

In 2004, Chen et al. presented a structure, a Conditional Outcome Preference Network (COP-net), for modeling preferences directly elicited from a user [2]. Chen et al. provides not only a qualitative representation of user preferences but also a quantitative representation by incorporating utility values into COP-nets. COP-nets consider uncertainty as well as the consistency issue on preference relations. COP-nets are expanded from CP-nets to indicate all preferences over all feasible outcomes. COP-nets are easy to check for consistency and redundancy and it is also easy to determine preferences over outcomes quickly by using transitively reduced COP-nets. COP-nets can handle preferences across different attributes. The main uses for COP-nets are to determine whether one outcome is preferred over another, and to estimate utilities for the entire set of outcomes. Based on a set of preferences from the user and the partial utility function u, specifying utilities for a subset O' of outcomes, an estimated utility function over the entire set O is produced.

Modeling COP-nets is very straightforward. A COP-net is a directed acyclic graph where if an outcome o_1 in the graph is an ancestor of another outcome o_2, then the user's true utility for o_1 is higher than that for o_2. Once the initial COP-net has been built, a consistency check is performed on the initial COP-net to ensure that it is acyclic, and then it is transitively reduced to obtain the reduced COP-net. This network is then used to

estimate the user's utilities for all outcomes for which the utility has not been elicited. From the result, Chen et al. found that a technique called the Longest-Path technique was the most accurate for predicting utilities, regardless of the numbers of attributes and conditional preferences [2].

Since COP-nets contain nodes for every possible outcome, if there are n attributes and each attribute has two possible values, then there are 2^n outcomes. To build a COP-network, all preferences that are specified by the user are then transformed to preference rules that are applied to all outcomes in the network. A directed edge from node n_i to node n_j is added if the preference $o_i \succ o_j$ holds for the corresponding outcomes.

After building an initial COP-network, Chen suggests how to create a reduced COP-network. For nodes n_i , n_j and n_k in an initial COP-network representing outcomes o_i , o_j and o_k, respectively, if there are two paths $n_i \rightarrow n_j \rightarrow n_k$ and $n_i \rightarrow n_k$, the second path is redundant and can be removed. After building a reduced COP-network, preferences can be checked as follows:

If n_i is a proper ancestor of n_j, o_i is preferred over o_j ($o_i \succ o_j$).

If n_i is a proper descendant of n_j, o_j is preferred over o_i ($o_j \succ o_i$)

If n_i is neither an ancestor nor a descendant of n_j, neither of o_i and o_j is known to be preferred over the other.

Also, a COP-network can be checked for consistency by ensuring that there is no outcome that is preferred over itself (no cycle in the COP-network).

We will apply the COP-net modeling process to the simple example in Figure 2.1. The process starts with extracting all feasible outcomes from the given preferences. In the simple example, three preferences are given: (1) $S_f \succ S_v$, (2) $S_v : W_r \succ W_w$, (3) S_f: $W_w \succ W_r$. All feasible outcomes are listed: $S_f \wedge W_r$, $S_f \wedge W_w$, $S_v \wedge W_r$, $S_v \wedge W_W$. Table 2.1 shows the corresponding node representation for all feasible outcomes.

Table 2.1: Node representation for outcomes

Outcomes	$S_f \wedge W_r$	$S_f \wedge W_w$	$S_v \wedge W_r$	$S_v \wedge W_W$
nodes	n_0	n_1	n_2	n_3

Next, we specify all *ceteris paribus* preferences from the given preference rules. By applying the preference rule (1) $S_f \succ S_v$, we get two preferences, $S_f \wedge W_r \succ S_v \wedge W_r$ and $S_f \wedge W_w \succ S_v \wedge W_w$ under the *ceteris paribus* assumption. Also, from preferences (2) and (3), it can be determined that $S_v \wedge W_r \succ S_v \wedge W_w$ and $S_f \wedge W_w \succ S_f \wedge W_r$. Therefore, finally we can get $n_0 \succ n_2$, $n_1 \succ n_3$, $n_2 \succ n_3$ and $n_1 \succ n_0$. The final step is to build an initial COP-net which is a directed graph. Figure 2.3 shows the initial COP-net from the simple example in Figure 2.1. Figure 2.3 (a) shows the induced preferences on all feasible outcomes and Figure 2.3 (b) shows the initial COP-net.

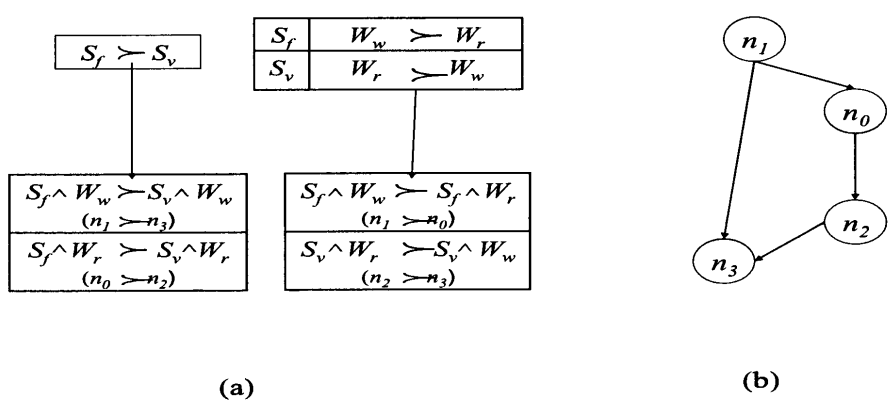


Figure 2.3: (a) the induced preferences for "My Dinner I"; (b) the initial COP-net

Next, we reduce the initial COP-net by removing any redundant arcs in the graph. Figure 2.4 (b) shows the reduced COP-net. The arc (n_1, n_3) from the preference $S_f \wedge W_w \succ S_v \wedge W_w$ has been removed.

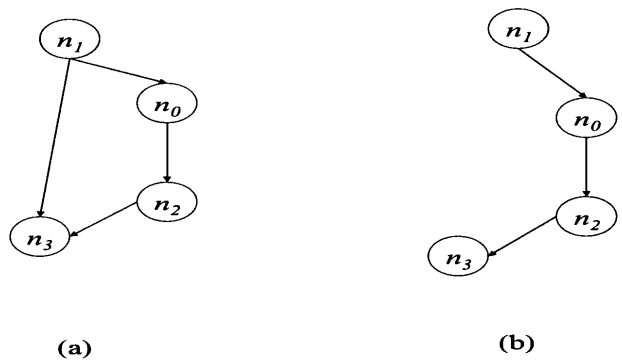


Figure 2.4: (a) the initial COP-net; (b) the reduced COP-net

Now, we extend this simple COP-net by adding a new attribute. The user adds a new attribute for bread = {roll, toast} with the preferences: the user prefers a roll if vegetable soup and red wine are served, and the user prefers toast if vegetable soup and white wine are served. In this example, five preferences are given: (1) $S_f \succ S_v$, (2) S_v: $W_r \succ W_w$, (3) S_f : $W_w \succ W_r$, (4) $S_v W_r : B_r \succ B_t$ and (5) $S_v W_w : B_t \succ B_r$. There is no preference between B_t and B_r whenever $S_f\, W_r$ and $S_f\, W_w$. In this example, we simplify $S_f \wedge W_r \wedge B_r$ to $S_f\, W_r\, B_r$ by removing the $\wedge$ symbol. So, all feasible outcomes are listed: $S_f W_w B_r$, $S_f W_w B_t$, $S_f W_r B_r$, $S_f W_r B_t$, $S_v W_w B_r$, $S_v W_w B_t$, $S_v W_r\, B_r$ and $S_v W_r B_t$. Table 2.2 shows the corresponding node representation for all feasible outcomes.

Table 2.2: Node representation for outcomes

Outcomes	$S_f W_w B_r$	$S_f W_w B_t$	$S_f W_r B_r$	$S_f W_r B_t$	$S_v W_w B_r$	$S_v W_w B_t$	$S_v W_r B_r$	$S_v W_r B_t$
Nodes	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7

Next, we specify all *ceteris paribus* preferences from the given preference rules. By applying the preference rule (1) $S_f \succ S_v$, we get four preferences, $S_fW_rB_r \succ S_vW_rB_r$, $S_fW_rB_t \succ S_vW_rB_t$, $S_fW_wB_r \succ S_vW_wB_r$ and $S_fW_wB_t \succ S_vW_wB_t$ under the *ceteris paribus* assumption. From the preference (2) S_v: $W_r \succ W_w$, we get two preferences, $S_vW_rB_r \succ S_vW_wB_r$ and $S_vW_rB_t \succ S_vW_wB_t$. From the preference (3) S_f: $W_w \succ W_r$, we get two preferences, $S_f W_w B_r \succ S_f W_r B_r$ and $S_f W_w B_t \succ S_f W_r B_t$. From the preferences (4) and (5), we get the preferences, $S_vW_rB_r \succ S_v W_rB_t$ and $S_vW_wB_t \succ S_vW_wB_r$. Therefore, finally we can get $n_0 \succ n_4$, $n_1 \succ n_5$, $n_2 \succ n_6$, $n_3 \succ n_7$, $n_0 \succ n_2$, $n_1 \succ n_3$, $n_6 \succ n_4$, $n_7 \succ n_5$, $n_6 \succ n_7$, and $n_5 \succ n_4$. The final step is to build an initial COP-net, which is a directed graph. Figure 2.5 shows the initial COP-net. Figure 2.5 (a) shows the induced preferences on all feasible outcomes and Figure 2.5 (b) shows the initial COP-net.

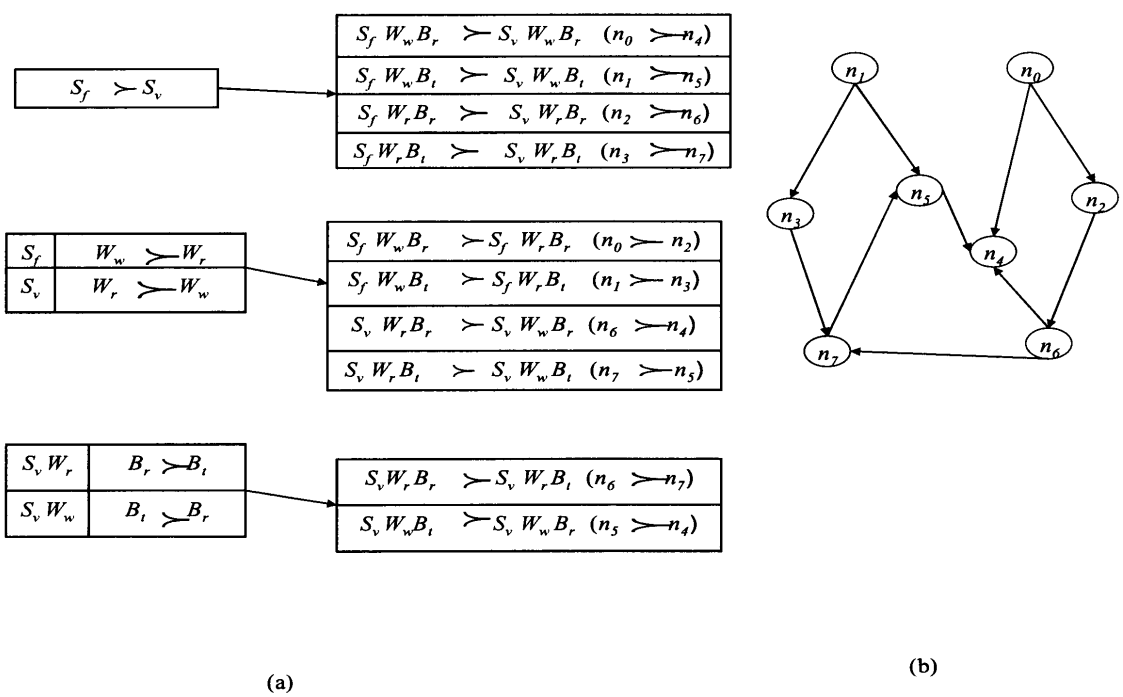


Figure 2.5: (a) the induced preferences; (b) the initial COP-net

Next, we reduce the initial COP-net by removing any redundant arcs in the graph. Figure 2.6 (b) shows the reduced COP-net. The arcs (n_0, n_4), (n_6, n_4) and (n_1, n_5) from the preferences, $S_fW_wB_r \succ S_vW_wB_r$, $S_vW_rB_r \succ S_vW_wB_r$ and $S_fW_wB_t \succ S_vW_wB_t$ have been removed.

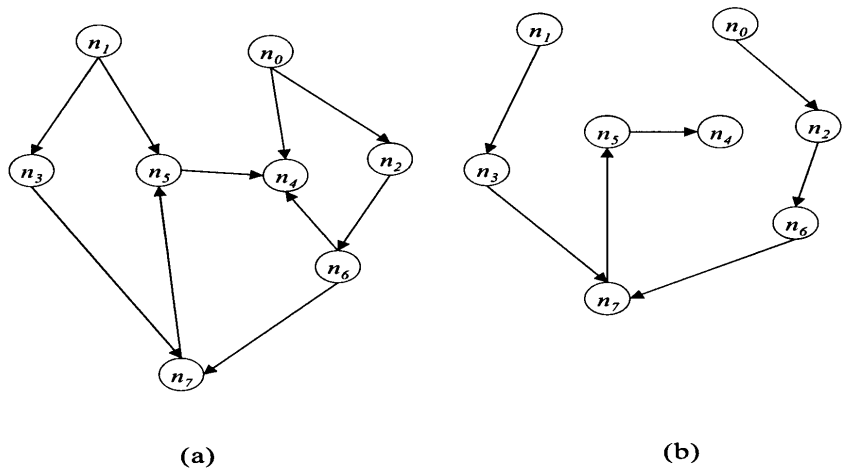


Figure 2.6: (a) the initial COP-net; (b) the reduced COP-net

In addition to obtaining a set of preference relations during the elicitation process, COP-nets also allow for querying about specific utilities for outcomes. Chen et al. suggest three different methods for estimating a utility function from a COP-net: the bounded method, the random-path method and the longest-path method. The Bounded method, first, sets upper and lower bounds for each outcome whose utility is unknown and assigns the average of these bounds as the utility value. The random-path method randomly selects a path of outcomes whose utilities are unknown and assigns utility to the outcomes in order. For example, if there is a path (o_1, o_2, o_3, o_4), then we know the outcome o_1 is the most preferable outcome and has the highest utility and o_4 is the least preferable outcome and has the lowest utility. We assign 1 to o_1 and 0 to o_4. Then, we assign utility to the other outcomes in order, and thus 0.67 to o_2 and 0.33 to o_3. Instead of randomly choosing a path, the longest path method chooses the longest acceptable

candidate path in the COP-net. A candidate path is only acceptable if it satisfies certain conditions that will ensure that the utility estimates over the entire network will be consistent; details can be found in [2]. The longest-path method was found to achieve the best results of the three. Figure 2.7 shows the resulting COP-net after assigning utilities to each outcome in the reduced COP-net in Figure 2.6 with the longest-path method. Initially, all top-level nodes (in this case, n_1 *and* n_0) are given a utility of 1, and all bottom-level nodes (n_4) a utility of 0. The longest path method then chooses the longest path such that the end paths have known or computed utility and the interior nodes do not. In this case the longest such path is (n_0, n_2, n_6, n_7, n_5, n_4) and utilities for interior nodes are then distributed evenly between 1 and 0 (i.e. 0.8, 0.6, 0.4, 0.2). The only remaining path after this is (n_1, n_3, n_7) giving n_3 utility equally between 1 and 0.4 (i.e. 0.7).

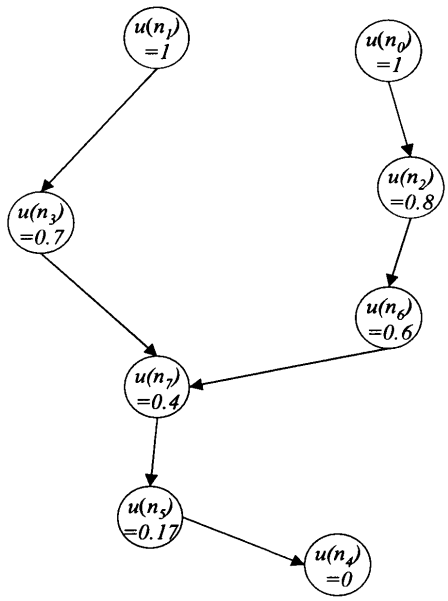


Figure 2.7: COP-net showing elicited and estimated utilities

2.4 Transitive Reduction Maintenance

In order to induce the reduced COP-net from the initial COP-net, Chen et al. apply transitive reduction to the initial COP-net to make it easy to calculate utility estimates using the longest path method. In order to maintain a COP-net after considering changing preferences, we need to develop an appropriate maintenance algorithm for the dynamic COP-net to find a transitive reduction more efficiently. We briefly start with basic definitions for general graphs. In Section 2.4.2, we will discuss basic concepts of transitive reduction and transitive closure. In Section 2.4.3, we will discuss some of the current transitive reduction maintenance algorithms in detail.

2.4.1 Graphs

A *graph* $G = (V, E)$ consists of a nonempty set V of vertices or nodes and a set E of edges that link the vertices together [12]. An edge e is a two-element subset of V. If an edge e is not an ordered set, then the graph G is called an *undirected graph*. If an edge e is an ordered set, then the graph G is called a *directed graph* and an edge e can be called an arc or directed edge. An arc $a = (x, y)$ is considered to be directed from x to y. A weighted graph will have a real-valued weight w associated with each edge or arc. Two vertices are *adjacent* if they are connected by the same edge.

A *path* is "a traversal of consecutive vertices along a sequence of edges" [12]. If a path leads from x to y, then y is called a *successor* of x and *reachable* from x, and x is called a *predecessor* of y. The length of a path is the number of edges or the sum of the weights assigned to the edges that are traversed along the path. A *directed path* is a path with

length greater than 1 in a directed graph. If there is no repeated edge in a directed path, then it is called a simple path. If the initial vertex is same as the final vertex in a path, then this path is called a *cycle*. A simple directed graph without cycles is called an acyclic directed graph.

2.4.2 Transitive reduction vs. transitive closure

Aho et al. in 1972 originally defined the transitive reduction compared to the transitive closure:

The transitive reduction G^R of a directed graph G is a minimal representation of the graph G such that the transitive closure of G^R is the same as the transitive closure of G, where the transitive closure $G^C=(V, E^C)$ of a graph G is a graph such that for all v, w in V, there is an edge(v, w) in E^C if and only if there is a directed path from v to w in G. [4]

We can check the existence of a path between two vertices in a graph by checking for the existence of a single edge in the transitive closure. Therefore, the transitive closure is often used to check minimum query time in database applications. Alternatively, the transitive reduction contains the minimum number of edges which satisfies the following properties: i) The vertex set of a directed graph $\boldsymbol{G}$ is equal to the vertex set of the transitive reduction $\boldsymbol{G^R}$. ii) There is a directed path from vertex u to vertex v in $\boldsymbol{G}$ if and only if there is a directed path from u to vertex v in $\boldsymbol{G^R}$. iii) There is no graph with fewer edges than $\boldsymbol{G^R}$ satisfying the above conditions [16].

2.4.3 Facilitating Transitive Reduction Maintenance

Since Aho et al. [4] originally defined the transitive reduction, there has not been much research on transitive reduction compared to the extensive research on transitive closure [16]. Some approaches to maintaining the transitive reduction have involved approximating the transitive reduction in a graph [12, 16]. Little research has been done with fully dynamic maintenance algorithms for transitive reduction [3, 16].

J.A. La Poutré and Jan Van Leeuwen, in 1988, presented an algorithm for the problem of efficiently updating the transitive closure and transitive reduction of a directed graph [11]. In this paper, they considered each insertion and deletion separately whenever an edge is inserted or deleted. Let $G=<V, E>$ represent a directed graph, $G^*=<V, E^*>$ its transitive closure and $G^-=<V, E^->$ its transitive reduction, and let E, E^* and E^- be represented by adjacency matrices. The adjacency matrix M of G is the $|V|\times|V|$ matrix with entries from $\{0, 1\}$ given by $M(i, j)$ being set to 1 iff $(i, j)\in E$ for $i, j\in V$. The adjacency matrix of the transitive closure G^* is denoted by M^* and the adjacency matrix of the transitive reduction G^- is denoted by M^-.

The algorithm is based on the input matrix N of G, where G is a directed graph. $N(k, m)$ is the number of edges incoming to m that are reachable from k. An edge(l, m) is said to be reachable from k if its starting node l is a successor of k. Figure 2.8 shows how edge(l, m) is reachable from a vertex k.

Figure 2.8: Reachability

Figure 2.9 shows an example of the input matrix N. In this example, there are 4 edges incoming to m and only 2 of the incoming edges are reachable from k. Therefore, the input matrix $N(k, m)$ has value 2.

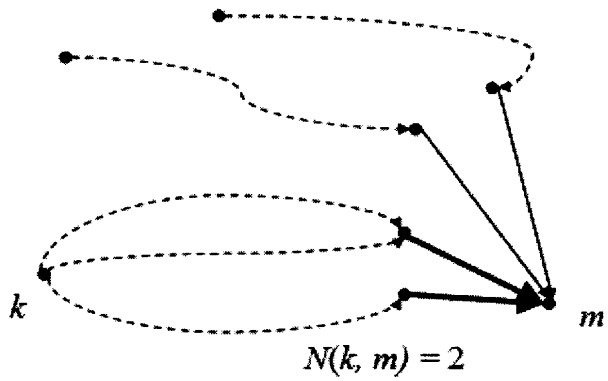


Figure 2.9: Input matrix example

A transitive reduction of an acyclic graph satisfies that $E^{-} = \{(k, m) \in V \times V \mid N(k, m) = M(k, m) = 1\}$. For an edge insertion, the first property is the following: If a new edge (i, j) is inserted in a directed graph G, then for $k, m \in V$, a vertex m is reachable from k if m is already a successor of k before an edge (i, j) insertion or if i is a successor of k and m is a successor of j. Figure 2.10 shows an example of the first property.

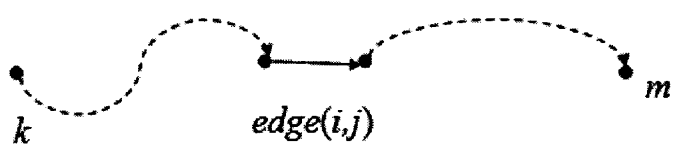


Figure 2.10: Reachable from k to m with new edge (i, j) insertion

There are two procedures to insert a new edge (i, j): the insert procedure and the adapt procedure. In the insert(i, j) procedure, there are two main processes. First, for each predecessor k of i, the input matrix $N(k, j)$ is increased by 1: $N(k, j) = N(k, j)+1$. After increasing the input matrix $N(k. j)$, if $N(k. j) = M(k, j) = 1$ then $M^{-}(k, j)$ is set to 1, otherwise $M^{-}(k, j)$ is set to 0. Second, for each vertex k that is a predecessor of i but not a

predecessor of j: $\{k \in V \mid M^*(k, i) = 1$ and $M^*(k, j) = 0\}$, the adapt(k) procedure is started after coloring j in red. In the adapt(k) procedure, it will adapt all affected successors of j after inserting an new edge (i, j). Let l be a red node. For each successor m of l, set input matrix $N(k, m) = N(k, m)+1$ and $M^-(k, m) = 0$. If $M^*(k, m) = 0$ then m is colored in red and the adapt(k) procedure is repeated until there is no red node. Therefore, the time complexity of an edge insertion in an acyclic graph for the insert procedure is $O(n)$, where n is the number of vertices. For the adapt procedure, the time complexity is $O(n.e_{new})$, where e_{new} is the number of edges after q edge insertions. The total time complexity of an edge insertion is $O(n^2 e_{new})$.

The delete algorithm is similar to the previous insert algorithm. There are two procedures for deleting an edge (i, j): the delete procedure and the adjust procedure. In the delete(i, j) procedure, there are two main processes. First, for each predecessor k of i, the input matrix $N(k, j)$ is decreased by 1: $N(k, j) = N(k, j)-1$. After decreasing the input matrix $N(k. j)$, if $N(k. j) = M(k, j)=1$ then $M^-(k, j)$ is set to 1, otherwise $M^-(k, j)$ is set to 0. Second, for each vertex k that is a predecessor of i but not a predecessor of j: $\{k \in V \mid M^*(k, i) = 1$ and $M^*(k, j)=0$ and $k \neq j\}$, the adjust(k) procedure is started after marking j. The adjust(k) procedure will adjust all affected successors of j after deleting an edge (i, j). Let m_0 be a marked node. $M^*(k, m_0)$ is set to 0 and unmark m_0. For each successor m^1 of m_0, set input matrix $N(k, m^1) = N(k, m^1)-1$. If $N(k, m^1)=0$ then m^1 is marked and if $N(k, m^1) = M(k, m^1) = 1$, then $M^-(k, m^1)$ is set to 1. Repeat the adjust(k) procedure until there is no marked node. Therefore, the time complexity of an edge deletion in an acyclic graph, for the delete procedure, is $O(n)$, where n is the number of vertices. The time complexity for the

adjust procedure is $O(n.e_{old})$, where e_{old} is the number of edges before q edge deletion. The total time complexity of an edge insertion is $O(n^2\ e_{old})$.

King and Sagert, in 2002, presented a fully dynamic algorithm for maintaining transitive closure using an adjacency matrix [3]. In this paper, the authors provided an algorithm with $O(n^2)$ time for each update and $O(1)$ time for each query, in directed acyclic graphs. Since transitive reduction is the inverse problem of transitive closure and has the same time complexity as transitive closure [4], King's algorithm has the same time complexity for the transitive reduction problem. The key idea of King's algorithm is to track the number of distinct directed paths from each vertex i to each vertex j in the graph, and for each insertion or deletion, add or subtract the number of directed paths between affected vertices. Let G be a directed acyclic graph containing vertices i, j, u, v. G is represented by the $n*n$ adjacency matrix M given by $M(i, j) = 1$ if *there is an edge(i, j) in G. Paths*(i, j) represents the number of distinct paths from i to j. If edge (u, v) is inserted (deleted) such that no cycles are created, the number of paths from i to j increases (decreases) by *Paths(i, u) * Paths(v, j),* where *Paths*(i, j) is the number of distinct directed paths from i to j in the graph. Figure 2.11 directly refers to King and Sagert's paper (2002) and it shows how inserting a single edge affects the number of paths between other vertices containing the updated edge.

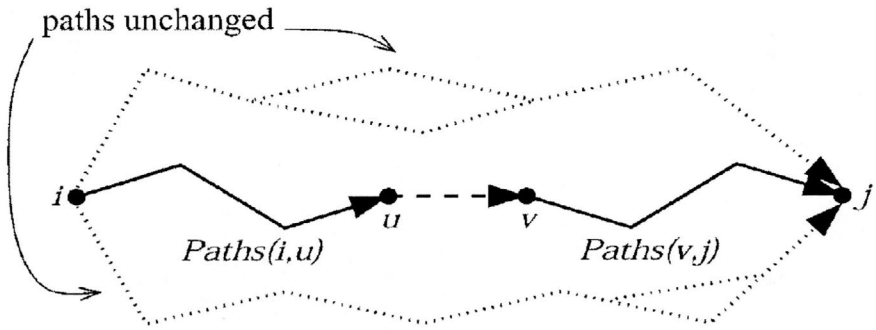


Figure 2.11: Inserting a single edge

The number of paths from i to j changes only if some path from i to j contains the updated edge (u, v) [3].

If edge (u, v) is inserted, it starts with finding all the vertices that have a path to vertex u and all the vertices that have a path from vertex v. Let the set of vertices that have a path to vertex u be ***Predecessors(u)*** and the set of vertices that have a path from vertex v be ***Successors***(v). For each vertex i in ***Predecessors***(u), the number of paths from i to v is increased by $Paths(i, u)$ (*i.e.* $Paths(i, v) = Paths(i, v) + Paths(i, u)$) and for each vertex j in ***Successors*** (v), $Paths(u, j)$ is increased by $Paths(v, j)$ (*i.e.* $Paths(u, j) = Paths(u, j) + Paths(v, j)$). Finally, for each vertex i in ***Predecessors***(u) and each vertex j in ***Successors***(v), $Paths(i, j)$ is increased by $Paths(i,u)*Paths(v,j)$ (*i.e.* $Paths(i,j) = Paths(i,j) + (Paths(i,u)*Paths(v,j))$). The process of an edge deletion is the same except that the number of paths is decreased. After maintaining the path matrix $Paths(i, j)$ to be equal to the number of distinct directed paths from i to j, the transitive closure of G is represented by the $n*n$ adjacency matrix M^* given by $M^*(i, j) = 1$ if $Paths(i, j) \neq 0$, and the transitive reduction of G is represented by the $n*n$ adjacency matrix M^+ given by $M^+(i, j) = 1$ if $Paths(i, j) = 1$ and $M(i, j) = 1$.

Figure 2.12 shows King's insert algorithm and Figure 2.13 shows King's delete algorithm for acyclic graphs.

```
Algorithm (Insert_Acyclic(E_v, G))
                    – to insert a set of edges incident to a common vertex v.

  for all i∈V do
    Predecessors(i) ←Σ_{(u,v) ∈E_v} Paths(i, u)
  for all j∈V do
    Successors(j) ←Σ_{(v,u) ∈E_v} Paths(u, j)
  for all i, j∈V\{v} such that Predecessors(i) ≠ 0 ∨ To(j) ≠ 0 do
    Paths(i, j) ← Paths(i, j) +[Paths(i, v) Successors(j)
                              + Predecessors(i) Paths(v, j)
                              + Predecessors(i) Successors(j)]
  j ← v
  for all i∈V\{v} such that Predecessors(i) ≠ 0 do
    Paths(i, j) ← Paths(i, j) + Predecessors(i)
  i ← v
  for all j∈V\{v} such that Successors(j)≠ 0 do
    Paths(i, j) ← Paths(i, j) + Successors(j)
  for all i, j∈V such that Paths(i, j) has increased from 0 do
    M*_G(i, j) ← 1
                                                    [King & Sagert, 2002]
```

Figure 2.12: King's insert algorithm

```
Algorithm (Delete_Acyclic(E_v, G))
                    - to delete a set of edges incident to a common vertex v.

  for all i∈V do such that Paths(i, u)≠ 0 do
   Preedecessors(i) ←Σ_{(u,v) ∈E_v} Paths(i, u)
  for all j∈V do such that Paths(u, j)≠ 0 do
    Successors(j) ←Σ_{(v,u) ∈E_v} Paths(u, j)
  j ← v
  for all i∈V\{v} such that Preedecessors(i) ≠ 0 do
    Paths(i, j) ← Paths(i, j) - Preedecessors(i)
  i ← v
  for all j∈V\{v} such that Successors(j)≠ 0 do
    Paths(i, j) ← Paths(i, j) - Successors(j)
  for all i, j∈V\{v} such that Preedecessors(i) ≠ 0 ∨ Successors(j) ≠ 0 do
    Paths(i, j) ← Paths(i, j) - [Paths(i, v) Successors(j)
              + Preedecessors(i) Paths(v, j)
              + Preedecessors(i) Successors(j)]
  for all i, j∈V such that Paths(i, j) has decreased to 0 do
    M*_G(i, j) ← 0
                                                    [King & Sagert, 2002]
```

Figure 2.13: King's delete algorithm

A deficiency of King's algorithm occurs when it computes the number of distinct directed paths, since it requires a great deal of processing time to update each affected cell in the path matrix. It is a main goal of this thesis to significantly reduce this processing time by proposing a method for maintaining transitive reduction/transitive closure without tracking the number of distinct directed paths.

3. Dynamic COP-nets

Since current approaches for representing preference relations have been designed based on the assumption that a user's preferences are stable, we have to rebuild the entire preference model whenever new preferences are added or current preferences are changed after building the initial preference model. Chen et al.[2] provided reduced COP-nets, corresponding to a transitive reduction of initial COP-nets, to reduce the redundancy of the initial network. It is desirable to provide a more flexible framework that can modify reduced COP-nets without altering or invalidating the existing preferences and without rebuilding the entire network. In this thesis, we present a new method that extends the previous COP-nets to represent changing preferences. We refer to the framework using the new method as a dynamic COP-net.

The other problem to be considered involves the need to be able to find a transitive reduction of the changing preference graph efficiently. According to Chen et al, transitively reducing the initial COP-net makes it easy to calculate utility estimates using the longest path method [2] as discussed in Section 2.3.2. The goal is to provide an efficient algorithm that maintains the transitive reduction in $O(n^2)$ time and can answer the question "Is outcome o_1 preferred over o_2?" in O(1) time. Although King's algorithm, as represented in Section 2.4.3, shows a worst-case time complexity of $O(n^2)$ time with O(1) query time, it still has some deficiencies in that it requires another path matrix for storing information that is not covered by the adjacency matrix. In order to store the number of paths between two vertices, a path matrix requires more time to manipulate than a binary matrix because it uses integer values instead of boolean entries. A goal of

this thesis is to provide a more efficient algorithm for the transitive reduction with maintaining a dynamic COP-net with $O(n^2)$ updating time and O(1) query time using only the adjacency matrix. We are not claiming to reduce the worst case complexity (i.e. it will remain at n^2), but we are still hoping for dramatic improvements in average time.

Since King's algorithm keeps track of the number of distinct directed paths between two vertices in the graph, it requires updating each affected cell in the path matrix. Therefore, the worst-case time complexity of King's algorithm is always $O(n^2)$ through its entire life time. Instead of tracking the number of distinct directed paths from i to j in the graph, our new approach will check if there is any directed path from i to j containing or not containing a modified arc e, in order to maintain the transitive reduction. Since our new algorithm will only keep track of the status between two vertices, once we reduce an arc, we do not need to update it later. Therefore, the processing time for maintaining the transitive reduction in our dynamic COP-nets would be decreased.

3.1 Representation of preferences with COP-nets

A dynamic COP-net follows the same preference representation rules used in COP-networks. A COP-network is a directed acyclic graph, with n vertices and m arcs. Each vertex represents an outcome and each directed arc(v_i, v_j) represents a preference relation between vertex v_i and vertex v_j meaning that a user prefers v_i to v_j . Therefore, n vertices represent a set of n outcomes. There exists a vertex v_1 with an in-degree of zero and another vertex v_n with out-degree of zero. Then the vertex v_1 represents the most preferred outcome and the vertex v_n represents the least preferred outcome [2]. As with

COP-nets, the structure of dynamic COP-nets allows preferences to be represented within attributes and across attributes by considering all feasible outcomes.

3.2 Algorithm for Maintaining Transitive Reduction

In order to provide efficient maintenance of the transitive reduction in $O(n^2)$ time with O(1) query time, we consider the fact that there are 4 basic conditions that can occur between two vertices i and j. If there is no *path* and no arc from i to j, then it is in condition 1. If there is an arc (i, j) but no other directed path, then it is in condition 2. If there is no arc but a directed path from i to j, then it is in condition 3. If there is an arc (i, j) and a directed path from i to j, then it is in condition 4. Figure 3-1 shows the 4 basic conditions between two vertices.

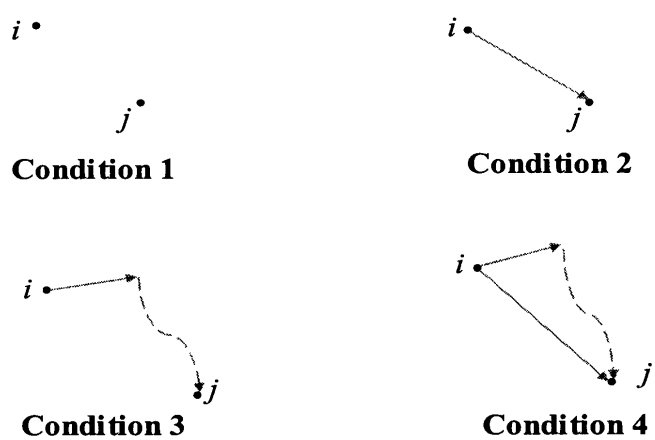


Figure 3-1 Four basic conditions between two vertices

In our method, we maintain two $n \times n$ binary matrices, M_1 and M_2, where n is the number of vertices in a COP-network, to represent these 4 conditions. Matrix M_1 stores the information about arcs representing a set of preferences elicited from a user. Therefore, if there is an arc(i, j) in a COP-net, then $M_1(i, j) = 1$. Matrix M_2 is used for a set of distinct

directed paths, and so if there is a directed path (of length greater than 1) from i to j in a COP-net, then $M_2(i, j) = 1$. Therefore, condition 1 can be represented with $M_1(i, j) = 0$ and $M_2(i, j) = 0$. Condition 2 can be represented with $M_1(i, j) = 1$ and $M_2(i, j) = 0$. Condition 3 can be represented with $M_1(i, j) = 0$ and $M_2(i, j) = 1$. Condition 4 can be represented with $M_1(i, j) = 1$ and $M_2(i, j) = 1$.

Let G=<V, E> be a directed acyclic graph representing a COP-network. The binary matrices M_1 and M_2 of G are |V| x |V| matrices with values {0, 1}.

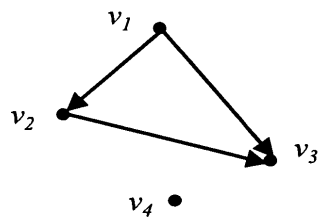
Building initial matrices M_1, M_2 is done as follows:

1. $M_1(i, j)$, $M_2(i, j) = 0$ for every $i, j \in V$.

2. Update $M_1(i, j) = 1$, if $(i, j) \in E$ for $i, j \in V$.

3. Add $M_2(i, j)=1$, if there is a directed path of length greater than 1 from i to j for $i, j \in V$.

Figure 3-2 shows an example of a dynamic COP-net with a set of vertices $V=\{v_1, v_2, v_3, v_4\}$ and a set of arcs $E=\{(v_1, v_2), (v_1, v_3), (v_2, v_3)\}$ and the 2 corresponding binary matrices M_1 and M_2.



Initial COP-network

	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	0	0	1	0
v_3	0	0	0	0
v_4	0	0	0	0

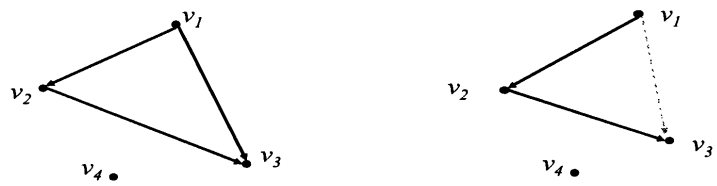
Matrix 1 (M_1)

	v_1	v_2	v_3	v_4
v_1	0	0	1	0
v_2	0	0	0	0
v_3	0	0	0	0
v_4	0	0	0	0

Matrix 2 (M_2)

Figure 3-2 Dynamic COP-net example 1

Finally, a *reduced* COP-network can be derived from Matrix 1 by removing redundant arcs, which can be done by converting $M_1(i, j)$ to 0, if $M_1(i, j)$=1 and $M_2(i, j)$=1. We define this operation as '-'. In Figure 3-3, the underlined values for $M_1(v_1, v_3)$ and $M_2(v_1, v_3)$ are both equal to 1, and so the value of the reduced matrix M_1-$M_2(v_1, v_3)$ is 0. Then the *reduced* COP-network G^R has a set of arcs $E^R = \{(v_1, v_2), (v_2, v_3)\}$, and thus the arc (v_1, v_3) has been removed. Figure 3-3 shows an initial COP-network and a reduced COP-network.



Initial COP-network G Reduced COP-network G^R

	v_1	v_2	v_3	v_4
v_1	0	**1**	**1**	0
v_2	0	0	**1**	0
v_3	0	0	0	0
v_4	0	0	0	0

Matrix 1 (M_1)

	v_1	v_2	v_3	v_4
v_1	0	0	**1**	0
v_2	0	0	0	0
v_3	0	0	0	0
v_4	0	0	0	0

Matrix 2 (M_2)

⇨

	v_1	v_2	v_3	v_4
v_1	0	**1**	0	0
v_2	0	0	**1**	0
v_3	0	0	0	0
v_4	0	0	0	0

Reduced Matrix (M_1 - M_2)

Figure 3-3 Dynamic COP-network with reduced COP-net

3.2.1 New preference insertion

Based on the 4 basic conditions, we develop the following three propositions.

Proposition 3.1 If there is any vertex v_i with sub-paths $p_1=(v_1, v_2, \ldots, v_i)$ and $p_2=(v_i, v_{i+1}, \ldots, v_n)$, then there is a path p from v_1 to v_n, $p=(v_1, v_2, \ldots, v_i, \ldots, v_n)$.

Proposition 3.2 If there is an arc (v_i, v_j) with sub-paths $p_1=(v_1, v_2, \ldots, v_i)$ and $p_2=(v_j, v_{j+1}, \ldots, v_n)$, then there is a path p from v_1 to v_n, $p=(v_1, v_2, \ldots, v_i, v_j, \ldots, v_n)$.

Proposition 3.3 If there is an arc (v_i, v_j) and a path $p_1=(v_j, v_{j+1}, \ldots, v_n)$, then there is a path p from v_i *to* v_n, $p=(v_i, v_j, \ldots, v_n)$.

Figure 3-4 shows Proposition 3.1, Proposition 3.2 and Proposition 3.3.

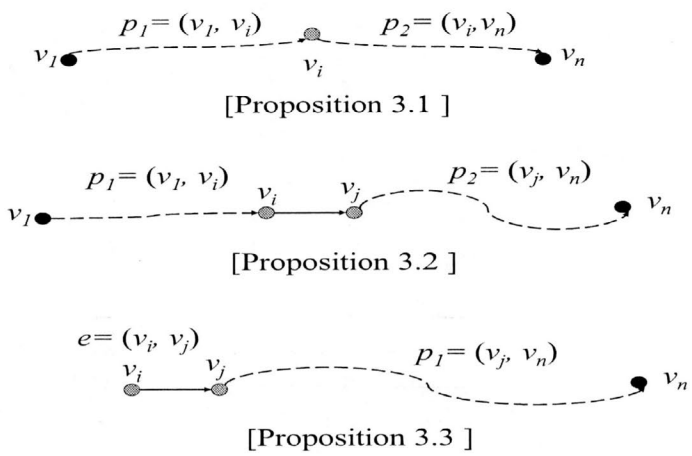


Figure 3-4 Proposition 3.1, Proposition 3.2 and Proposition 3.3

From these three propositions, we can develop our algorithm for a new preference insertion. Since a preference is represented by an arc in a dynamic COP-network, we consider inserting a new arc (v_i, v_j) to the initial COP-network when there is a new preference added such that a user prefers an outcome v_i to an outcome v_j. Since a new arc (v_i, v_j) is inserted, for all vertices I that have a path to v_i and all vertices J that have a path from v_j, then, by Proposition 3.2 with new arc (v_i, v_j), there is now a directed path from I to J. Therefore, $M_2(I, J)=1$.

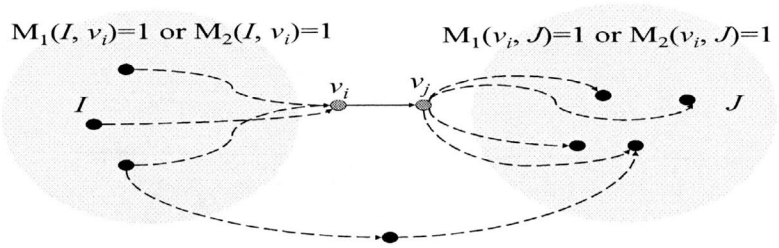


Figure 3-5 Insertion of a new arc (v_i, v_j)

Therefore, when inserting a new arc (v_i, v_j) in a directed acyclic graph, there are three steps. The first step is: for all $I \in V$ such that $M_1(I, v_i) = 1$ or $M_2(I, v_i) = 1$, if $M_2(I, v_j) = 0$, then $M_2(I, v_j) = 1$. The second step is: for all $J \in V$ such that $M_1(v_j, J) = 1$ or $M_2(v_j, J) = 1$, if $M_2(v_i, J) = 0$, then $M_2(v_i, J) = 1$. The third step is: for all $I, J \in V$ from step 1 and step 2, if $M_2(I, J) = 0$, then $M_2(I, J) = 1$. Finally, the fourth step is to find the reduced matrix M_1-M_2: for all $i, j \in V$, if $M_1(i, j) = 1$ and $M_2(i, j) = 0$, then M_1-$M_2(i, j) = 1$; otherwise M_1-$M_2(i, j) = 0$. Figure 3-6 shows the new insertion algorithm for a dynamic COP-network.

Algorithm (Insert $((v_i, v_j), G)$)
– *to insert a single edge* (v_i, v_j) to G.

$M_1(v_i, v_j) \leftarrow 1$

for all $I \in V$ such that $M_1(I, v_i) = 1$ or $M_2(I, v_i) = 1$ do – *Step 1*
if $M_2(I, v_j) = 0$, *then* $M_2(I, v_j) = 1$

for all $J \in V$ such that $M_1(v_j, J) = 1$ or $M_2(v_j, J) = 1$ do – *Step 2*
if $M_2(v_i, J) = 0$, *then* $M_2(v_i, J) = 1$

for all I, J from *Step* 1 and *Step* 2 – *Step 3*
if $M_2(I, J) = 0$, *then* $M_2(I, J) = 1$

return $M_1 - M_2$ – *Step 4*
[Dynamic COP-net]

Figure 3-6 Insertion algorithm for dynamic COP-nets

Figure 3-7 shows two binary matrices M_1 and M_2 in Step 1 of inserting the new arc (v_3, v_4), using the example of a dynamic COP-network from Figure 3-3. The circled region in M_1 shows that $M_1(v_1, v_3)$ and $M_1(v_2, v_3)$ have the values of 1 and therefore $I = \{v_1, v_2\}$. The circled region in M_2 shows that $M_2(v_1, v_4)$ and $M_2(v_2, v_4)$ now have the values of 1 after Step 1.

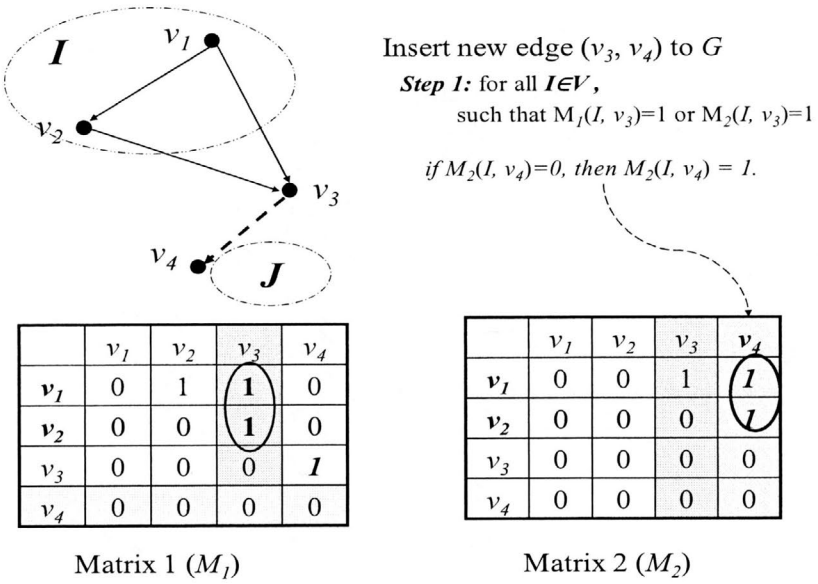


Figure 3-7 Insertion algorithm: Step 1

Since there are no vertices J such that $M_1(v_4, J)=1$ or $M_2(v_4, J)=1$, there is no update from Step 2 and Step 3. Now, we will look at another example. Figure 3-8 shows an initial network G_1, its reduced COP-network $G^R{}_1$ and two initial matrices M_1 and M_2.

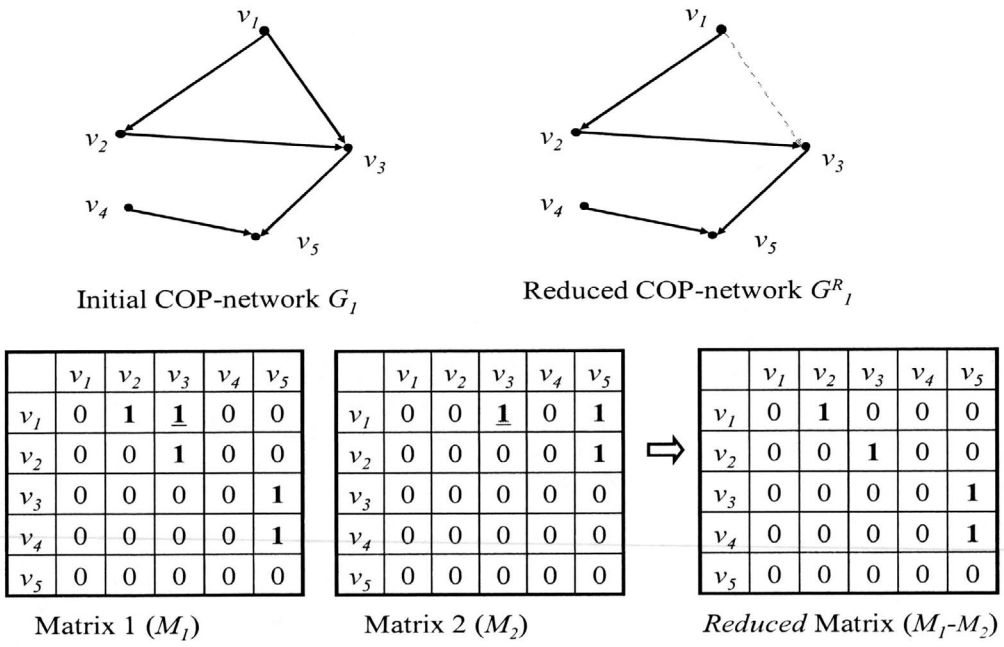
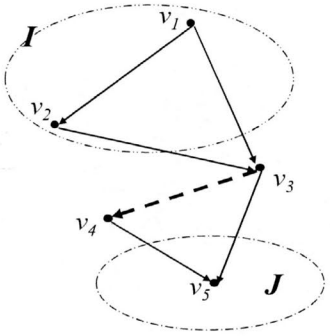


Figure 3-8 Dynamic COP-network: Example 2

Figure 3-9 shows two binary matrices M_1 and M_2 in Step 1 of inserting the new arc (v_3, v_4), using the example of the COP-network G_1 in Figure 3-8.



Insert new edge (v_3, v_4) to G_1

Step 1: for all $I \in V$,

such that $M_1(I, v_3)=1$ or $M_2(I, v_3)=1$

if $M_2(I, v_4)=0$, *then* $M_2(I, v_4) = 1$.

	v_1	v_2	v_3	v_4	v_5
v_1	0	**1**	**1**	0	0
v_2	0	0	**1**	0	0
v_3	0	0	0	***1***	**1**
v_4	0	0	0	0	**1**
v_5	0	0	0	0	0

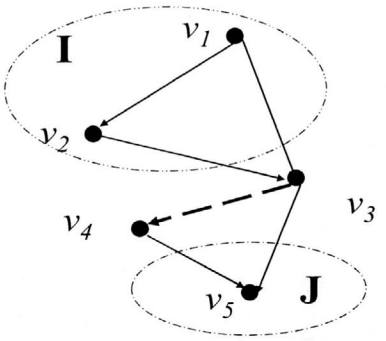
Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	**1**	**1**	**1**
v_2	0	0	0	**1**	**1**
v_3	0	0	0	0	0
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)

Figure 3-9 Insertion algorithm (Example 2): Step 1

Figure 3-10 shows two binary matrices M_1 and M_2 in Step 2 of inserting the new arc (v_3, v_4).



Insert new edge (v_3, v_4) to G_1

Step 2: for all $J \in V$,

such that $M_1(v_4, J)=1$ or $M_2(v_4, J)=1$

if $M_2(v_3, J)=0$, *then* $M_2(v_3, J) = 1$.

	v_1	v_2	v_3	v_4	v_5
v_1	0	**1**	**1**	0	0
v_2	0	0	**1**	0	0
v_3	0	0	0	***1***	**1**
v_4	0	0	0	0	**1**
v_5	0	0	0	0	0

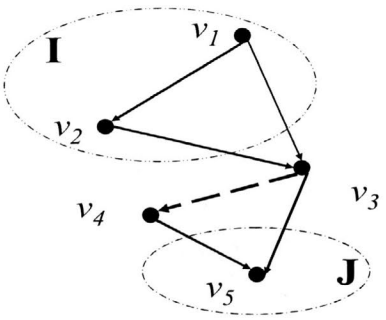
Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	**1**	**1**	**1**
v_2	0	0	0	**1**	**1**
v_3	0	0	0	0	***1***
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)

Figure 3-10 Insertion algorithm (Example 2): Step 2

Figure 3-11 shows two binary matrices M_1 and M_2 in Step 3 of inserting the new arc (v_3, v_4). In this step 3, there is no update for $M_2(I, J)$, since there is no I, J such that $M_2(I, J)=0$.



Insert new edge (v_3, v_4) to G_1

Step 3: for all ***I*** and ***J*** from *step* 1 and *step* 2,

if $M_2(I, J)=0$, *then* $M_2(I, J) = 1$.

	v_1	v_2	v_3	v_4	v_5
v_1	0	**1**	**1**	0	0
v_2	0	0	**1**	0	0
v_3	0	0	0	***1***	**1**
v_4	0	0	0	0	**1**
v_5	0	0	0	0	0

Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	**1**	**1**	**1**
v_2	0	0	0	**1**	**1**
v_3	0	0	0	0	**1**
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)

Figure 3-11 Insertion algorithm (Example 2): Step 3

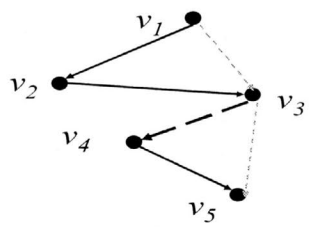
Finally, Figure 3-12 shows the reduced COP-network from the two finalized matrices M_1 and M_2. We can find the reduced COP-network from the minus operation M_1 - M_2.

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	**1**	0	0
v_2	0	0	1	0	0
v_3	0	0	0	***1***	**1**
v_4	0	0	0	0	1
v_5	0	0	0	0	0

Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	**1**	1	1
v_2	0	0	0	1	1
v_3	0	0	0	0	**1**
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)



Reduced COP-net

	v_1	v_2	v_3	v_4	v_5
v_1	0	**1**	0	0	0
v_2	0	0	**1**	0	0
v_3	0	0	0	**1**	0
v_4	0	0	0	0	**1**
v_5	0	0	0	0	0

Reduced Matrix (M_1-M_2)

Figure 3-12 Dynamic COP-network with Reduced COP-net

3.2.2 Preference deletion

In our dynamic COP-network, we also provide a method for maintaining a reduced COP-network when *deleting* an arc. A user may change his/her preferences because of a change in the preference itself, because of a change in the variables used for preference expression or because of a change in the preference domain [10]. For example, a user's financial environment may change and after increasing his/her income, the user is no longer concerned about the price of car. In this case, the preference domain changes to remove the price attribute from the domain. In another case, a user simply changes his/her preference over a car. The user now prefers a black color for a car whereas the user preferred white before. In this case, we need to change the user's preference relation. We make this change by deleting the previous preference and adding a new preference.

We first delete the user's previous preference: a user prefers white color to black color. Then, we add a new preference: a user prefers black color to white color.

To maintain a reduced COP-network with an arc (v_i, v_j) deleted, we have to answer the question "Is there a directed path from a vertex v_1 to a vertex v_n not containing the arc (v_i, v_j) ?". We call this question the sensitivity query. To answer the sensitivity query, we develop Proposition 3.4 and Proposition 3.5 based on the previous Propositions 3.1, 3.2 and 3.3.

Proposition 3.4 Let there be two directed paths $p_1=(v_1, v_2, \ldots, v_k, \ldots, v_{i-1}, v_i)$ and $p_2=(v_j, v_{j+1}, \ldots, v_{j+m}, \ldots, v_{n-1}, v_n)$. If there is an arc (v_k, v_{j+m}), then there is a path p from v_1 to v_n, including an arc (v_k, v_{j+m}), namely $p=(v_1, v_2, \ldots, v_k, v_{j+m}, \ldots, v_n)$.

Figure 3-13 shows Proposition 3.4.

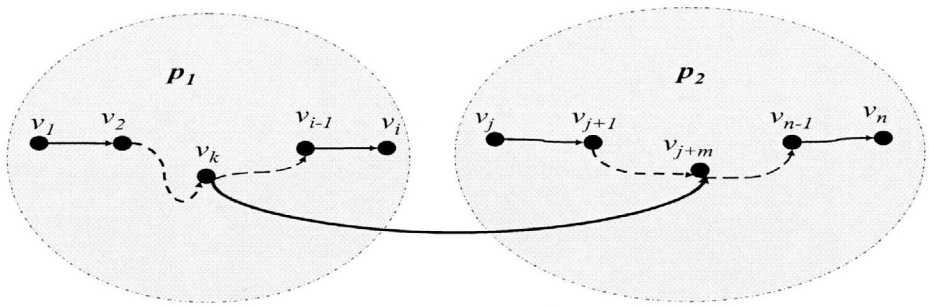


Figure 3-13 Proposition 3.4

Proposition 3.5 Let there be two directed paths $p_1=(v_1, v_2, \ldots, v_k, \ldots, v_{i-1}, v_i)$ and $p_2=(v_j, v_{j+1}, \ldots, v_{j+m}, \ldots, v_{n-1}, v_n)$. If there is a *directed path* from a vertex v_k in p_1 to a vertex v_{j+m} in

p_2, then there is *path p* from v_1 *to* v_n, including two vertices v_k and v_{j+m}, namely $p=(v_1, v_2, \ldots, v_k, \ldots, v_{j+m}, \ldots, v_n)$. Figure 3-14 shows Proposition 3.5.

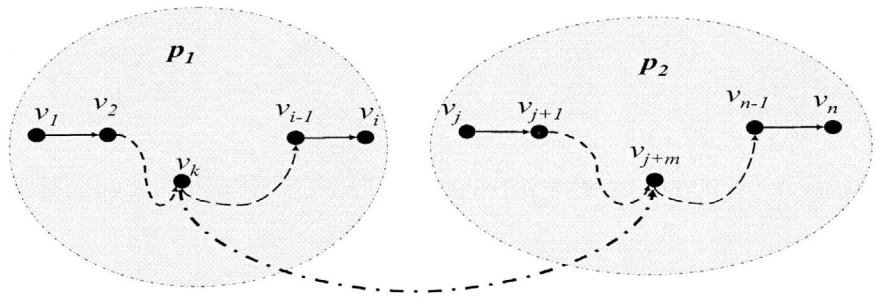


Figure 3-14 Proposition 3.5

From these Propositions 3.4 and 3.5, we can answer the sensitivity query "Is there a directed path from a vertex v_1 to a vertex v_n not containing the arc (v_i, v_j) ?". If there is any path from a vertex in a path from v_1 to v_i to a vertex in a path from v_j to v_n (where those vertices are not v_i and v_j), then the answer is "yes" which means that there is a directed path from a vertex v_1 to a vertex v_n not containing the arc (v_i, v_j). This is important since it indicates that there will still be a path from v_1 to v_n after the removal of (v_i, v_j).

We develop our deletion algorithm based on the 4 basic conditions. For the deletion of an arc (v_i, v_j) in a dynamic COP-net, the first step is to find all vertices in the reduced COP-net that will be affected by removing the arc (v_i, v_j): Define the set of vertices I to include v_i and all ancestors of v_i. Define the set J to include v_j and all descendants of v_j. Second, mark the pairs of vertices that are involved in any directed path from I to J not containing

the arc (v_i, v_j), by applying Proposition 3.3 to find if there are paths from vertices in I to vertices in J not containing the arc (v_i, v_j), as follows:

Step 2: Define the sets $I_1 = \{i \in I \mid \exists k \in V \setminus I$ such that $M_1(i, k) = 1\}$ and $K = \{k \in V \setminus I \mid \exists i \in I$ such that $M_1(i, k) = 1\}$.

For each $i_1 \in I_1$:

Step 2-1 Define the set $I_2 = \{i_2 \in I \mid \exists i_1 \in I_1$ such that $M_1(i_2, i_1) = 1$ or $M_2(i_2, i_1) = 1\}$ and $K_1 = \{k \in K \mid$ such that $M_1(i_1, k) = 1\}$. If $k \in K_1 \cap J$ and $M_2(i_2, k)$ is unmarked, then mark $M_2(i_2, k)$.

Step 2-2 Define the set $J_2 = \{j_2 \in J \mid \exists k \in K_1$ such that $M_1(k, j_2) = 1$ or $M_2(k, j_2) = 1\}$. For all $j_2 \in J_2$ such that $M_2(i_1, j_2)$ is unmarked, mark $M_2(i_1, j_2)$.

Step 2-3, for all $i_2 \in I_2$ and $j_2 \in J_2$ from step 2-1 and step 2-2, if $M_2(i_2, j_2) = 1$ and unmarked, then mark $M_2(i_2, j_2)$.

Figure 3-15 depicts these sets graphically for a given $i_1 \in I_1$.

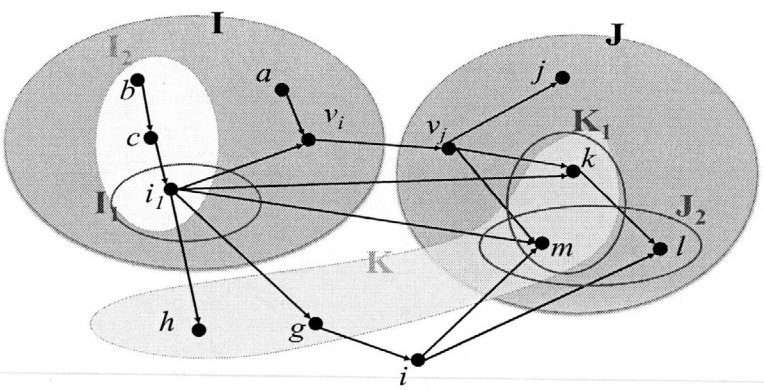


Figure 3-15 Graphical Sets Information for a given $i_1 \in I_1$

Third, for all $i \in I$ and $j \in J$ which are not marked in Step 2, if $M_2(i, j) = 1$, then let $M_2(i, j) = 0$. Figure 3-16 shows the new deletion algorithm for a dynamic COP-net.

Algorithm (Delete $((v_i, v_j), G)$)

– *to delete* ***a single edge*** (v_i, v_j) to G.

If $M_1(v_i, v_j) = 1$, *then* $M_1(v_i, v_j) = 0$

If $M_2(v_i, v_j) = 0$, *then* do

***Step* 1.** Define the set of vertices $I = \{i \in V$ such that $M_1(i, v_i) = 1$ or $M_2(i, v_i) = 1$ or $i = v_i\}$ and $J = \{j \in V$ such that $M_1(v_j, j) = 1$ or $M_2(v_j, j) = 1$ or $j = v_j\}$

***Step* 2.** Define the sets $I_1 = \{i_1 \in I \mid \exists\, k \in V \setminus I$ such that $M_1(i, k) = 1\}$ and $K = \{k \in V \setminus I \mid \exists\, i \in I$ such that $M_1(i, k) = 1\}$.

For each $i_1 \in I_1$:

Step 2-1 Define the set $I_2 = \{i_2 \in I \mid \exists\, i_1 \in I_1$ such that $M_1(i_2, i_1) = 1$ or $M_2(i_2, i_1) = 1\}$ and $K_1 = \{k \in K \mid$ such that $M_1(i_1, k) = 1\}$. If $k \in K_1 \cap J$ and $M_2(i_2, k)$ is unmarked, then mark $M_2(i_2, k)$. $- I_2$

Step 2-2 Define the set $J_2 = \{j_2 \in J \mid \exists\, k \in K$ such that $M_1(k, i_2) = 1$ or $M_2(k, i_2) = 1\}$. For all $i_1 \in I_1, j_2 \in J_2$ such that $M_2(i_1, j_2)$ is unmarked, then mark $M_2(i_1, j_2)$. $- J_2$

Step 2-3, for all $i_2 \in I_2$ and $j_2 \in J_2$ from step 2-1 and step 2-2, if $M_2(i_2, j_2) = 1$ and unmarked, then mark $M_2(i_2, j_2)$.

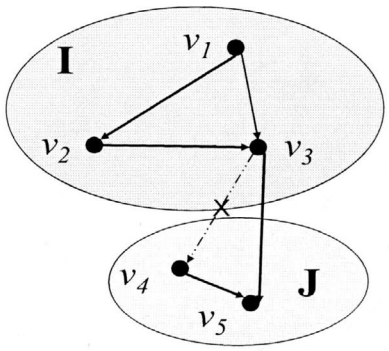
***Step* 3.** for all $i \in I, j \in J$ which are not marked from the step 2, if $M_2(i, j) = 1$, then $M_2(i, j) = 0$.

***Step* 4.** return $M_1 - M_2$

[Dynamic COP-net]

Figure 3-16 Deletion algorithm for Dynamic COP-nets

Figure 3-17 shows two binary matrices M_1 and M_2 in Step 1 of deleting an arc (v_3, v_4) in the dynamic COP-network (Example 2) from Figure 3-8. The shaded columns in M_1 and M_2 are used to find the set I of predecessors of v_3, and the shaded rows in M_1 and M_2 are to find the set J of successors of v_4. The circled regions in M_1 show that $M_1(v_1, v_3)$ and $M_1(v_2, v_3)$ have the values of 1 resulting in $I = \{v_1, v_2, v_3\}$ and that $M_1(v_4, v_5)$ has the value of 1 resulting in $J = \{v_4, v_5\}$. From Step 1, $I = \{v_1, v_2, v_3\}$ and $J = \{v_4, v_5\}$.



Delete an edge (v_3, v_4) from G_1

Step 1: Define the set of vertices $I=\{i \in V$ such that $M_1(i, v_i)=1$ or $M_2(i, v_i)=1$ or $i=v_i\}$ and $J=\{j \in V$ such that $M_1(v_j, j)=1$ or $M_2(v_j, j)=1$ or $j=v_j\}$

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	0	0	1	0	0
v_3	0	0	0	0	1
v_4	0	0	0	0	1
v_5	0	0	0	0	0

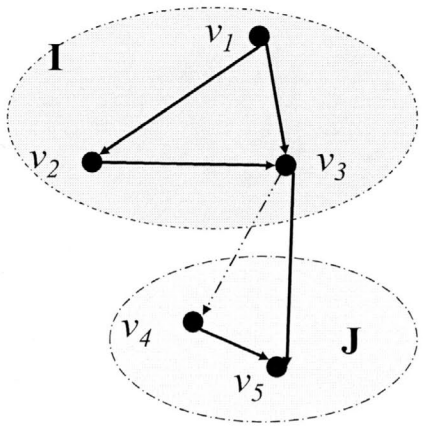
Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	1	1	1
v_2	0	0	0	1	1
v_3	0	0	0	0	0
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)

Figure 3-17 Deletion algorithm (Example 2): Step 1

Figure 3-18 shows two binary matrices M_1 and M_2 in Step 2 of deleting an arc (v_3, v_4). In this Step 2, we define $I_1=\{v_3\}$. The dashed circle in M_1 shows that M_1 (v_3, v_5) has the value of 1 and so $K=\{v_5\}$. In Step 2-1, for a given $i_1 \in I_1$ where, in this case, $i_1= v_3$, we define the set $I_2=\{v_1, v_2\}$ (since v_1 and v_2 have 1 in the column for v_3 in M_1 and M_2). Then, we mark two cells in the matrix M_2: $M_2(v_1, v_5)$ and $M_2(v_2, v_5)$. Since there is no vertex set J_2 in Step 2-2, there is no process in Step 2-3. The circles in M_1 show the sets $I_1=\{v_3\}$, $I_2=\{v_1, v_2\}$, $K=\{v_5\}$.



Delete an edge (v_3, v_4) from G_1

– *to delete a single edge* (v_3, v_4) to G.

Step 2. Define the sets $I_1=\{i_1 \in I \mid \exists\, k \in V \setminus I$ such that $M_1(i, k)=1\}$ and $K=\{k \in V \setminus I \mid \exists i \in I$ such that $M_1(i, k)=1\}$.

For each $i_1 \in I_1$:

Step 2-1 Define the set $I_2=\{i_2 \in I \mid \exists i_1 \in I_1$ such that $M_1(i_2, i_1)=1$ or $M_2(i_2, i_1)=1\}$ and $K_1=\{k \in K \mid$ such that $M_1(i_1, k)=1\}$. If $k \in K_1 \cap J$ and $M_2(i_2, k)$ is unmarked, then mark $M_2(i_2, k)$. $-I_2$

Step 2-2 Define the set $J_2=\{j_2 \in J \mid \exists\, k \in K$ such that $M_1(k, i_2)=1$ or $M_2(k, i_2)=1\}$. For all $i_1 \in I_1$, $j_2 \in J_2$ such that $M_2(i_1, j_2)$ is unmarked, then mark $M_2(i_1, j_2)$. $-J_2$

Step 2-3, for all $i_2 \in I_2$ and $j_2 \in J_2$ from step 2-1 and step 2-2, if $M_2(i_2, j_2)=1$ and unmarked, then mark $M_2(i_2, j_2)$.

Marked in Step2-1

		v_1	v_2	v_3 (I_1)	v_4	v_5 (K)
I_2	v_1	0	1	***1***	0	0
	v_2	0	0	***1***	0	0
I_1	v_3	0	0	0	**0**	***1***
	v_4	0	0	0	0	1
	v_5	0	0	0	0	0

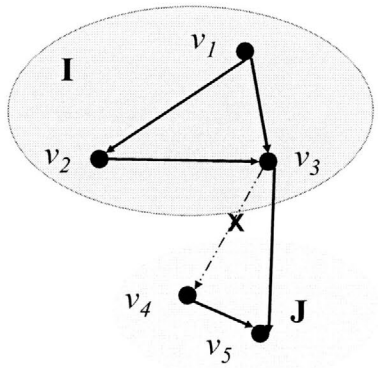
Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	***1***	1	
v_2	0	0	0	1	
v_3	0	0	0	0	0
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)

Figure 3-18 Deletion algorithm (Example 2): Step 2

Figure 3-19 shows two binary matrices M_1 and M_2 in Step 3 of the deleting an arc (v_3, v_4). In this Step 3, we update two cells in the matrix M_2: $M_2(v_1, v_4)=0$ and $M_2(v_2, v_4)=0$ that are not marked.



Delete an edge (v_3, v_4) from G_1

Step 3: for all $i \in I$ and $j \in J$ which are not marked from the step 2, *if* $M_2(i, j)=1$, *then* $M_2(i, j) = 0$.

				J	
	v_1	v_2	v_3	v_4	v_5
I v_1	0	1	1	0	0
v_2	0	0	1	0	0
v_3	0	0	0	0	1
v_4	0	0	0	0	1
v_5	0	0	0	0	0

Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	1	0	
v_2	0	0	0	0	
v_3	0	0	0	0	0
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)

Figure 3-19 Deletion algorithm (Example 2): Step 3

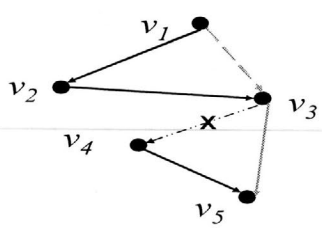
Finally, Figure 3-20 shows the reduced COP-network from the two finalized matrices M_1 and M_2. We can find the reduced COP-network from the operation M_1 - M_2.

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	0	0	1	0	0
v_3	0	0	0	0	1
v_4	0	0	0	0	1
v_5	0	0	0	0	0

Matrix 1 (M_1)

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	1	0	1
v_2	0	0	0	0	1
v_3	0	0	0	0	0
v_4	0	0	0	0	0
v_5	0	0	0	0	0

Matrix 2 (M_2)



Reduced COP-net

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	0	0
v_2	0	0	1	0	0
v_3	0	0	0	0	1
v_4	0	0	0	0	1
v_5	0	0	0	0	0

Reduced Map (M_1-M_2)

Figure 3-20 Dynamic COP-network with Reduced COP-net

4. Evaluation and Analysis

The main goal of the work described in this thesis is to provide a faster algorithm than King's existing algorithm for maintaining the transitive reduction of dynamic COP-nets, as a user's preferences change. We will implement our new algorithm, as well as King's algorithm, and we will compare their running times for maintaining the transitive reduction in order to verify the main goal. If our new algorithm is more efficient than King's existing algorithm, we will determine why.

4.1 Evaluation

In this section, we will discuss how the proposed method compares to King's algorithm and to a method involving COP-net reconstruction, in terms of efficiency. We will compare the average running time taken to build an initial preference network, as well as the time taken to update a single new preference. We will also compare the maximum number of updates (inserts/deletes) that can be performed within a given time. We will discuss evaluation plans that were developed and examine the results that were obtained.

4.1.1 Evaluation Plan

There are two main purposes to our evaluation. Since the main purpose of the proposed algorithm is to maintain a reduced COP-network with changing preferences, we first compare the efficiency of reduced COP-networks and unreduced COP-networks, when it comes to the running time of the utility estimation algorithm. Second, we examine how efficient the proposed method is, compared to King's algorithm and a method involving

COP-net reconstruction. In this step, we examine three different categories of experiments. When the number of attributes is small, the average running time (insert/delete) with King's algorithm and the proposed algorithm are both around 0ms. Therefore, we start with at least 5 attributes. We will first examine the average running time of building an initial preference network with different numbers of attributes (5, 6, 7 and 8 attributes) and different numbers of preferences, using the proposed algorithm and King's algorithm. Second, we will examine the average running time of updates (inserts/deletes) with the proposed algorithm and King's algorithm, and compare these to the time required to reconstruct a COP-net from scratch after the insertion/deletion of a preference. Also, if the number of attributes is greater than 8, the running time is significantly increased with King's algorithm and so we will examine this situation with a third approach. We will examine the percentage of the time in which construction can be completed within a given time, using King's algorithm and the proposed method.

We test four different sizes of graphs based on different numbers of attributes (n=5, 6, 7 and 8 attributes), with each attribute having two possible values. Thus the number of outcomes is 2^n. In order to test different densities for each graph, we choose five different values for the number of initial preferences generated: 2^n, $2^n(n+1)$, $2^n(4n+1)$, $2^n(9n+1)$ *and* $min\{2^n(16n+1), 2^{2n}\}$. Once we generate initial preferences, we remove any redundant preferences and inconsistent preferences. Therefore, the number of edges in an initial graph is never more and typically much less than the number of initial preferences.

For each set of test conditions, we randomly generate 100 different initial sets of preferences, to build an initial graph, and a set of 100 new preferences (updates) for each graph. This provides us with a total of 10,000 test cases for each set of conditions.

4.1.2 Reduced COP-networks vs. Unreduced COP-networks

The first experiment tests the average running time required find the longest path in each network. This is important since the utility estimation algorithm used relies on repeatedly finding the longest path between pairs of vertices. Thus, faster longest path identification implies faster utility estimation. Table 4.1 and Figure 4.1 show the results of comparing the average running time of finding the longest path in a reduced network to the average running time of finding the longest path in an unreduced network, based on different numbers of attributes (5, 6, 7 and 8) using the Depth-first search algorithm. It shows that finding the longest path to estimate utilities can be done more quickly in reduced COP-networks than in unreduced COP-networks, for larger numbers of attributes. This serves as justification for our desire to maintain reduced networks.

Table 4.1 Average time of finding longest path (ms)

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Unreduced network	Reduced network
5	299	165.76	34.12	0.15	0.10
6	1,408	732.14	71.07	0.23	0.08
7	5,504	2,826.66	122.57	0.62	0.23
8	12,544	6,552.08	310.15	1.31	0.73

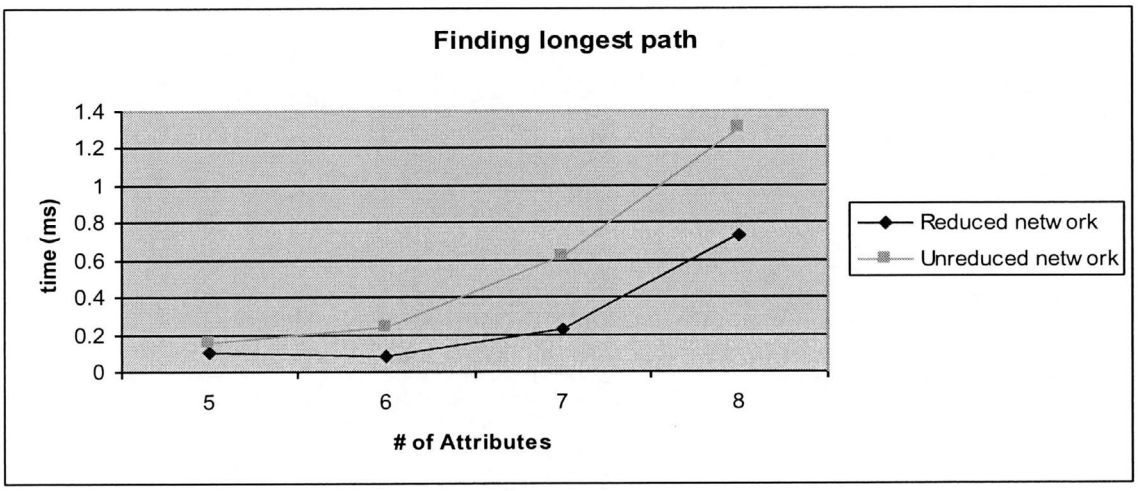


Figure 4.1 Finding longest path in reduced and unreduced networks

4.1.3 Proposed vs. Current methods

In this experiment, we will examine how the efficiency of the proposed method compares to that of King's algorithm and a method involving COP-net reconstruction. There are three ways in which we examine its efficiency. The first way is by comparing the initial construction times, the second is by comparing the average update time (insert/delete) of the three different methods and the third one is by comparing the maximum number of inserts/deletes using the three methods.

4.1.3.1 Initial Construction time

Table 4.2 and Figure 4.2 show the average initial construction times with 8 attributes and different densities by changing the number of preferences. The number of edges in an initial graph and the number of edges in a reduced graph show the different densities with 8 attributes. For example, when the number of initial preferences is 256, the average number of edges in the initial (sparse) graph is 253.26 and the average number of edges in the reduced graph is 247.27, and thus the reduction process results in the removal of

very few edges. Also, when the number of initial preferences is 33,024, the average number of edges in the initial (dense) graph is 16,717.04 and the average number of edges in the reduced graph is 295.28, showing that many edges are removed during the reduction process. The results demonstrate that the proposed method is significantly more efficient than King's method, especially for dense graphs.

Table 4.3 and Figure 4.3 show the average initial construction times with different numbers of attributes. The number of preferences shows the average number of preferences tested for each different number of attributes. The results demonstrate again that the proposed method outperforms King's method. The difference in performance increases as the number of attributes grows.

Table 4.4 and Figure 4.4 show the initial construction times with *sparse* graphs by changing the number of attributes. Table 4.5 and Figure 4.5 show the initial construction times with *dense* graphs by changing the number of attributes. Once again, the results show that King's algorithm takes more time to build an initial graph in all situations and that this is particularly noticeable for dense graphs and for larger numbers of attributes.

Table 4.2 Initial construction time with different densities

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Ratio (King/Proposed)
8	256	253.26	247.27	13.14	350.33	26.66
8	2,304	1,505.77	355.90	205.59	5,168.13	25.14
8	8,448	4,603.99	334.24	577.62	21,409.19	37.06
8	18,688	9,680.36	318.08	1,031.95	48,112.87	46.62
8	33,024	16,717.04	295.28	1,819.95	79,999.05	43.96

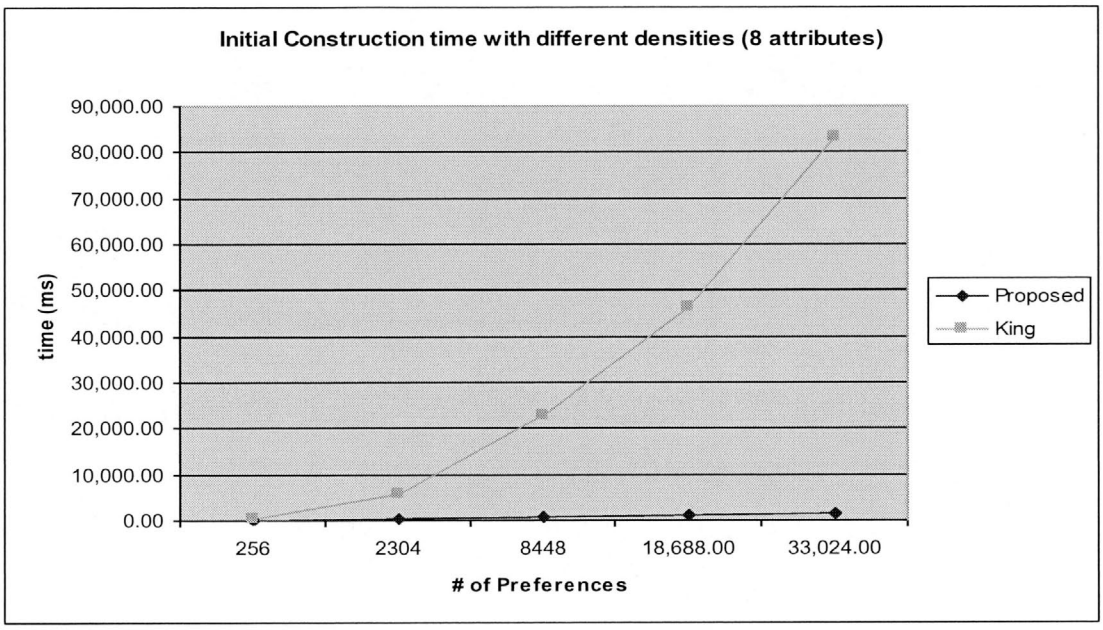


Figure 4.2 Initial construction time with different densities

Table 4.3 Initial construction time with different numbers of attributes

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Ratio (King/Proposed)
5	299	165.76	34.12	2.67	7.35	2.75
6	1,408	732.14	71.07	14.07	153.11	10.88
7	5,504	2,826.66	122.57	114.99	2,209.34	19.21
8	12,544	6,552.08	310.15	729.65	31,007.91	42.50

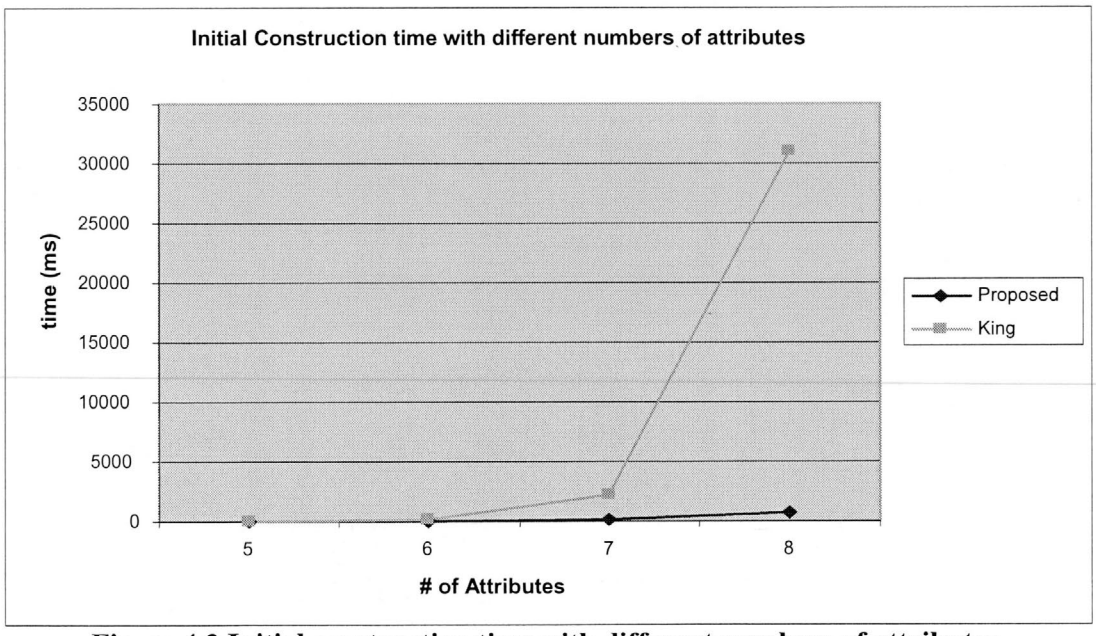


Figure 4.3 Initial construction time with different numbers of attributes

Table 4.4 Initial construction time for sparse graphs

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Ratio (King/Proposed)
5	32	29.99	28.24	0.16	0.78	4.88
6	64	61.93	58.8	1.27	5.71	4.50
7	128	125.65	1.21	3.56	38.92	10.93
8	256	253.26	247.27	13.14	350.33	26.66

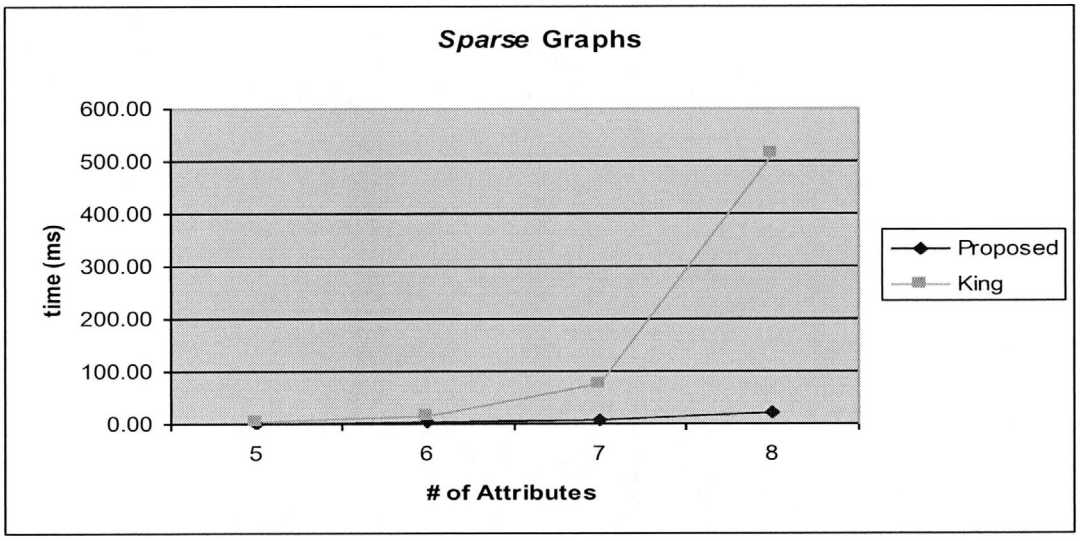


Figure 4.4 Initial construction time for sparse graphs

Table 4.5 Initial construction time for dense graphs

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Ratio (King/Proposed)
5	672	341.66	34.84	5.48	16.05	2.93
6	3,520	1713.44	66.19	34.35	380.58	11.08
7	14,464	7,120.98	132.80	291.52	5,223.43	17.92
8	33,024	16,717.04	295.28	1,819.95	79,999.05	43.96

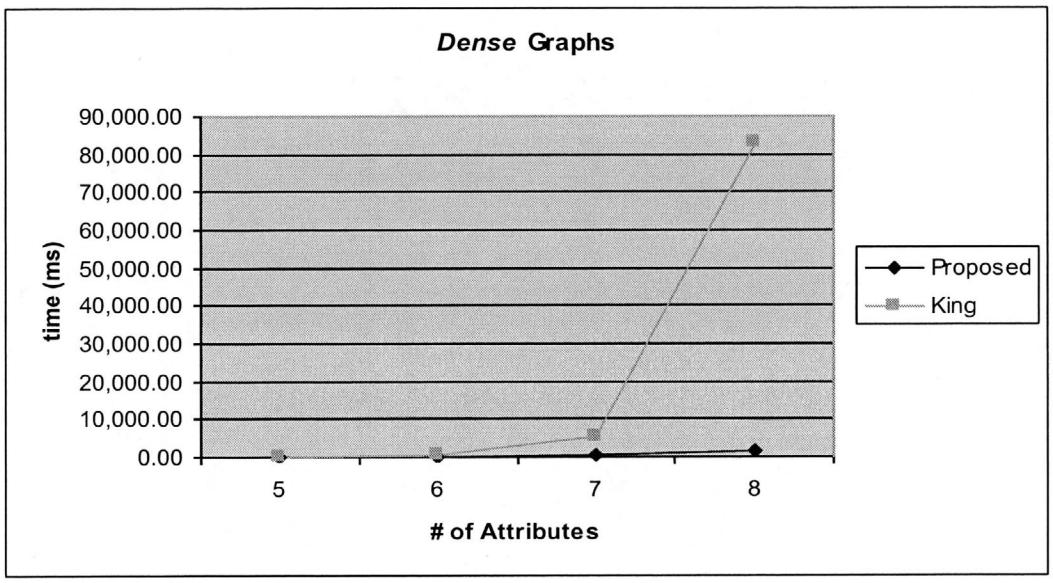


Figure 4.5 Initial construction time for dense graphs

4.1.3.2 Average update time

In this set of experiments, we compare the average time required for inserting and deleting preferences, using three different approaches: the proposed method, King's method, and a technique that involves reconstructing the graph from scratch after the insertion/deletion of a preference.

Table 4.6 and Figure 4.6 show the average insertion time with different density graphs using 8 attributes. Table 4.7 and Figure 4.7 show the average insertion time for different numbers of attributes. Table 4.8 and Figure 4.8 show the average insertion time for *sparse* graphs using different numbers of attributes. Table 4.9 and Figure 4.9 show the average insertion time for *dense* graphs using different numbers of attributes. Each set of results shows that the insertion time with the proposed method is more efficient for a given density and that it is not greatly affected by the number of attributes, whereas the running time of inserting new preferences using King's algorithm increases when the

number of attributes increases, and reconstruction time increases with increasing density and increasing number of attributes.

Table 4.6 Average insertion time with different densities (8 attributes)

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
8	256	253.24	246.78	0.15	1.76	0.51
8	2,304	1,500.86	356.77	0.12	1.93	0.61
8	8,448	4,609.93	336.24	0.10	2.03	0.83
8	18,688	9,681.63	315.27	0.06	1.89	1.03
8	33,024	16,721.48	295.04	0.05	1.71	1.29

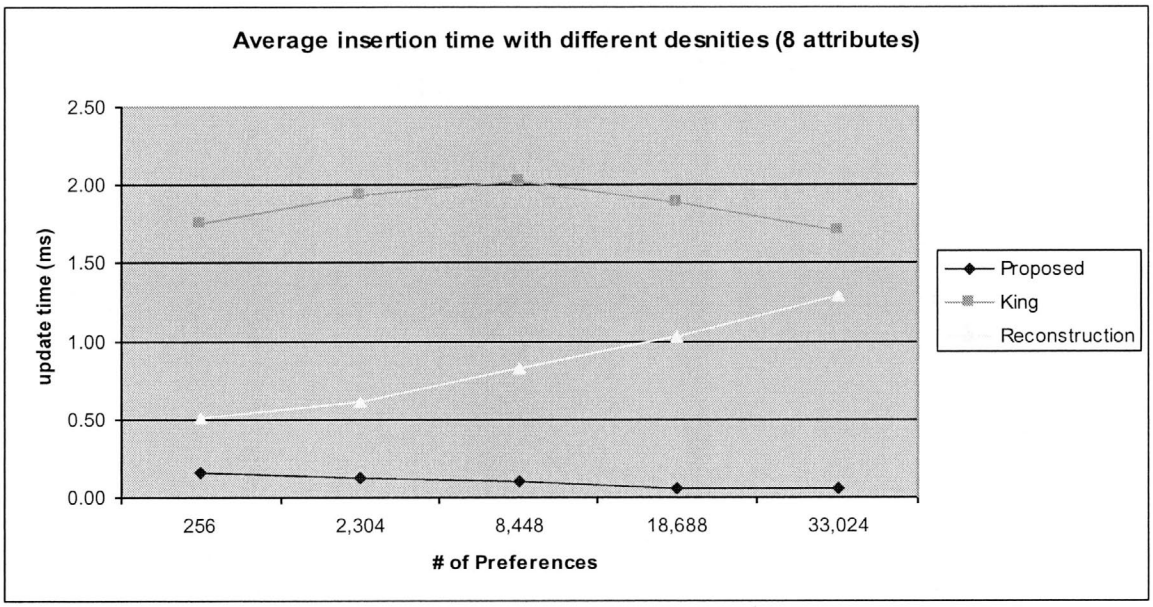


Figure 4.6 Average insertion time with different density graphs

Table 4.7 Average insertion time with different numbers of attributes

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
5	299	165.25	33.90	0.04	0.04	0.06
6	1,408	732.36	70.94	0.03	0.11	0.15
7	5,504	2,824.98	147.43	0.04	0.35	0.28
8	12,544	6,553.43	310.02	0.10	1.86	0.85

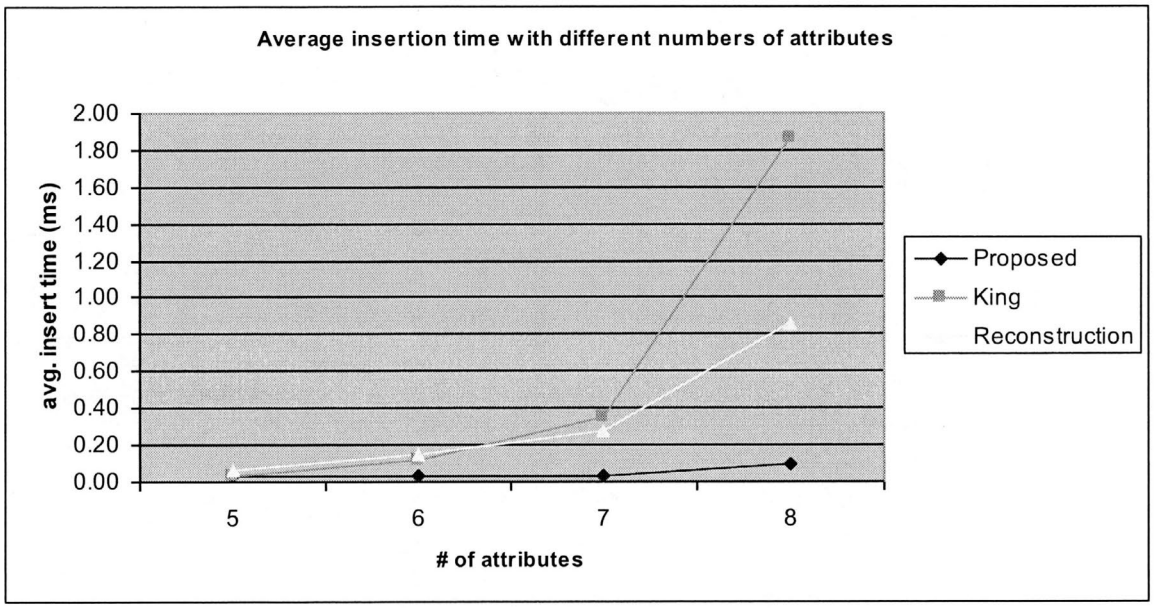


Figure 4.7 Average insertion time with different numbers of attributes

Table 4.8 Average insertion time with sparse graphs

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
5	32	30.05	28.33	0.07	0.06	0.06
6	64	62.01	58.94	0.06	0.18	0.08
7	128	125.68	121.25	0.06	0.53	0.23
8	256	253.24	246.78	0.15	1.76	0.51

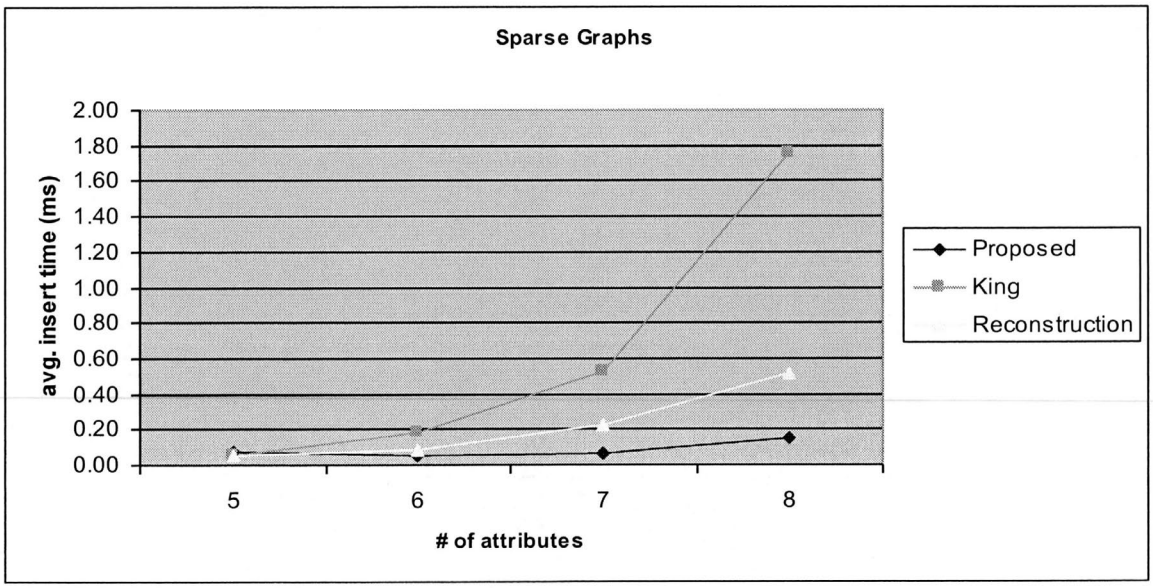


Figure 4.8 Average insertion time with sparse graphs

Table 4.9 Average insertion time with dense graphs

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
5	672	339.63	34.28	0.01	0.01	0.05
6	3,520	1,714.41	66.34	0.02	0.09	0.27
7	14,464	7,119.28	132.71	0.03	0.22	0.35
8	33,024	16,721.48	295.04	0.05	1.71	1.29

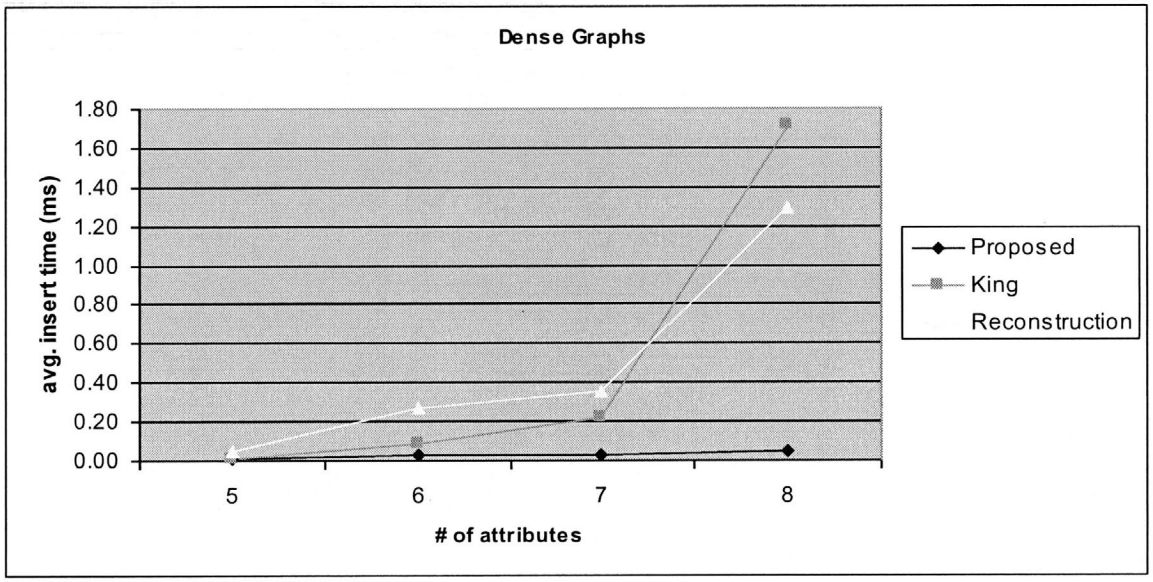


Figure 4.9 Average insertion time with dense graphs

Table 4.10 and Figure 4.10 show the average deletion time with different density graphs using 8 attributes. Table 4.11 and Figure 4.11 show the average deletion time for different numbers of attributes. Table 4.12 and Figure 4.12 show the average deletion time for *sparse* graphs using different numbers of attributes. Table 4.13 and Figure 4.13 show the average deletion time for *dense* graphs using different numbers of attributes. Each set of results shows that the reconstruction time is less than maintaining a reduced COP-network using either King's deletion algorithm or the proposed deletion algorithm. However, the reconstruction method still requires that the consistency of the graph be checked, since in order to find the longest path to estimate utilities we have to ensure that

the graph is acyclic. With the consistency condition, the proposed method is more efficient with large numbers of attributes except in the case of dense graphs. For dense graphs or a small number of attributes, King's method is always more efficient than the proposed method.

Table 4.10 Average deletion time with different densities (8 attributes)

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
8	256	253.26	247.27	0.40	3.99	0.38
8	2,304	1,505.77	355.90	2.13	3.11	0.46
8	8,448	4,603.99	334.24	2.40	2.93	0.65
8	18,688	9,680.36	318.08	2.53	2.63	0.88
8	33,024	16,717.04	295.28	2.69	2.52	1.21

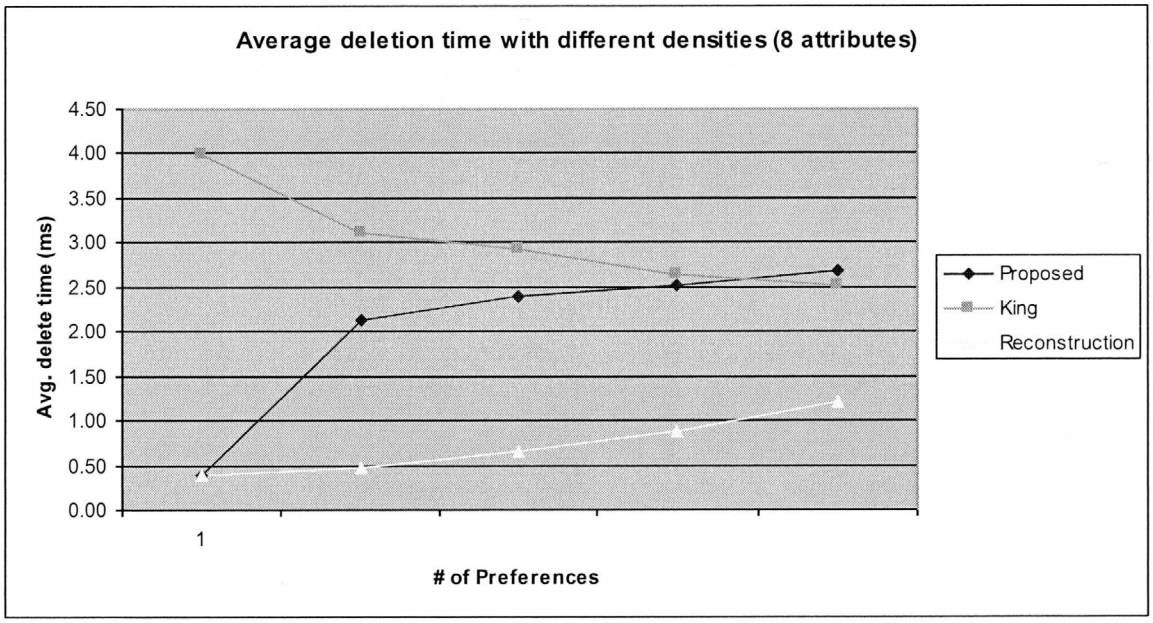


Figure 4.10 Average deletion time with different density graphs

Table 4.11 Average deletion time with different numbers of attributes

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
5	299	165.76	34.12	0.08	0.04	0.03
6	1,408	732.14	71.07	0.12	0.15	0.05
7	5,504	2,826.66	122.57	0.45	0.54	0.20
8	12,544	6,552.08	310.15	2.03	3.03	0.72

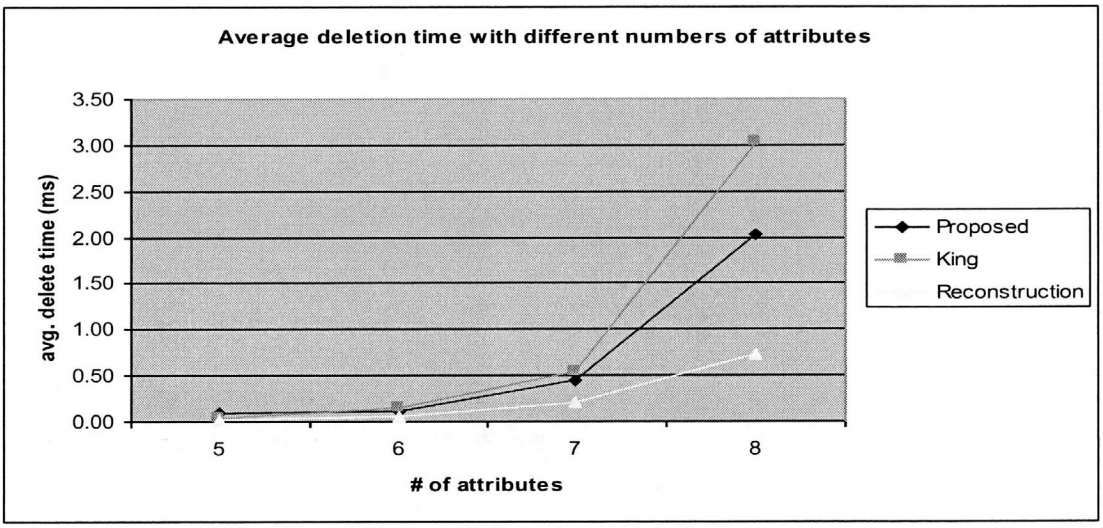


Figure 4.11 Average deletion time with different numbers of attributes

Table 4.12 Average deletion time with sparse graphs

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
5	32	29.99	28.24	0.10	0.06	0.02
6	64	61.93	58.8	0.10	0.23	0.04
7	128	125.65	1.21	0.21	0.77	0.09
8	256	253.26	247.27	0.40	3.99	0.38

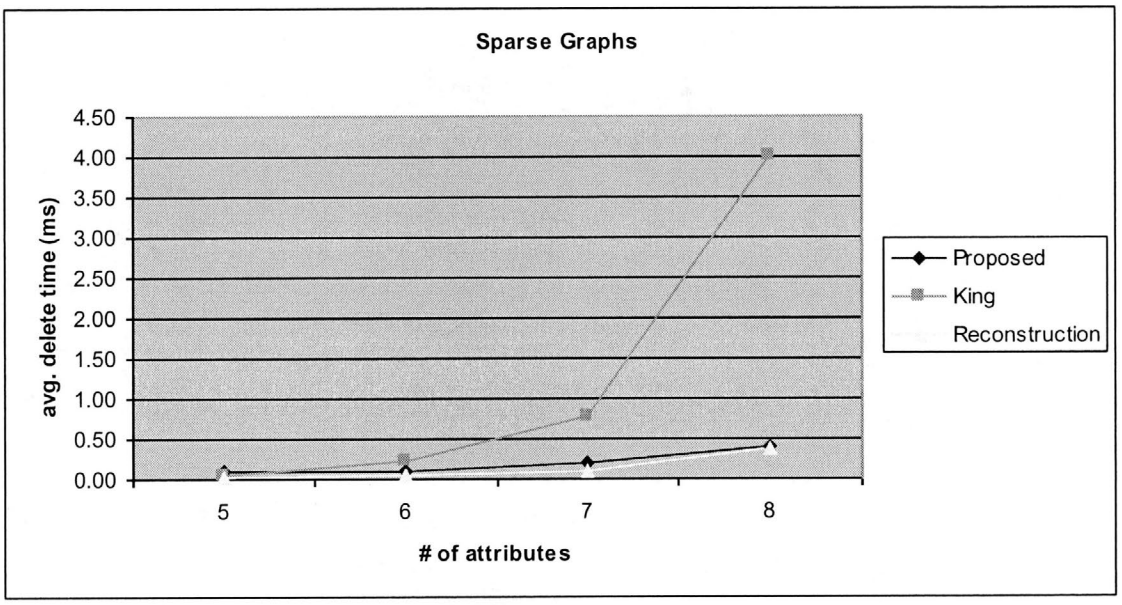


Figure 4.12 Average deletion time with sparse graphs

Table 4.13 Average deletion time with dense graphs

# of Attributes	# of Preferences	Initial graph # of edges	Reduced graph # of edges	Proposed Method time (ms)	King's Method time (ms)	Reconstruction time (ms)
5	672	341.66	34.84	0.05	0.04	0.03
6	3,520	1,713.44	66.19	0.14	0.12	0.07
7	14,464	7,120.98	132.80	0.59	0.46	0.31
8	33,024	16,717.04	295.28	2.69	2.52	1.21

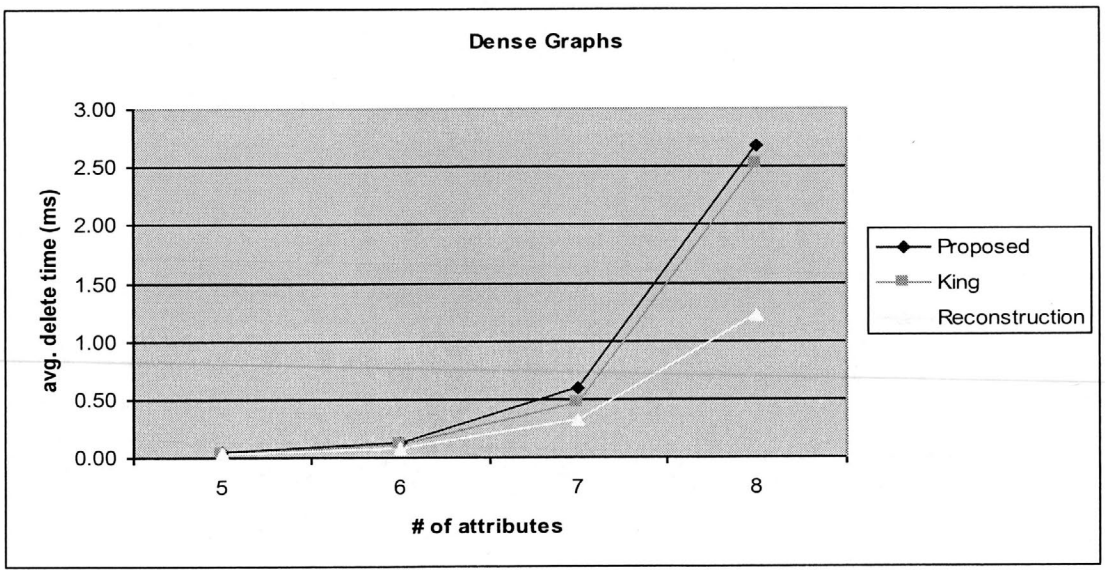


Figure 4.13 Average deletion time with dense graphs

4.1.3.3 Construction success rate within a given time limit

Since the running time with a number of attributes greater than 8 can be unreasonably long, in this experiment, we will examine how often the initial construction of a graph with a large number of attributes can be completed in a given limited amount of time (500ms). In this experiment, we can compare the proposed method to King's method. Table 4.14 and Figure 4.14 show the results of this experiment. The proposed method is able to complete the initial construction within the 500ms time limit at a significantly higher rate than King's method.

Table 4.14 Construction success rate with different # of attributes in 500ms

# of Attributes	# of Preferences	Initial graph (# of edges)	Proposed Method (%)	King's Method (%)
9	512	508.00	100.00	6.09
10	1,024	1,021.00	66.61	0.77
11	2,048	2,041.00	17.02	0.10
12	4,096	4,095.00	3.85	0.02
13	8,192	8,187.00	0.96	0.00
14	16,384	16,378.00	0.19	0.00

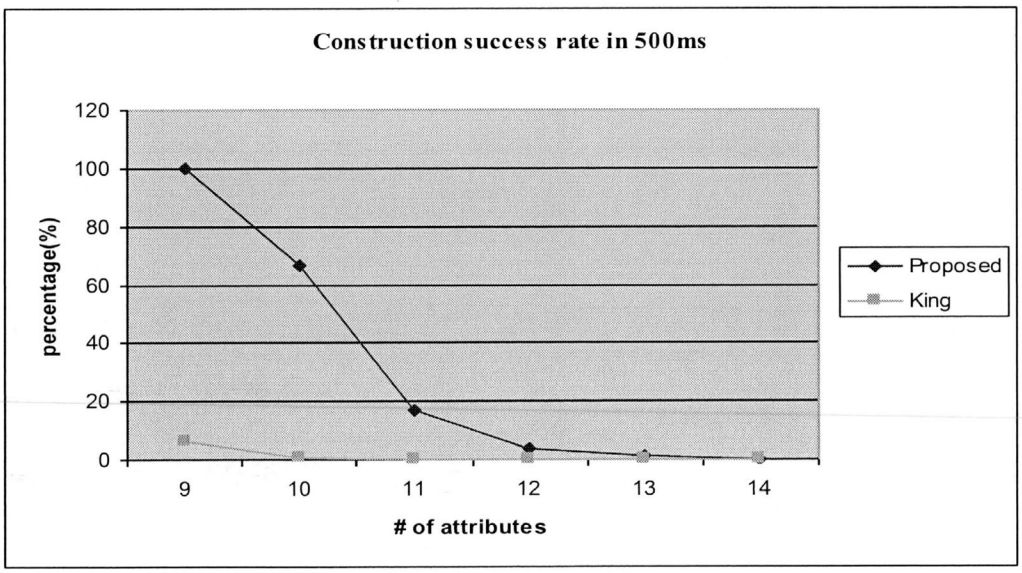


Figure 4.14 Construction success rate with different # of attributes in 500ms

4.1.3.4 Average number of updates with large number of attributes in a given time

In this experiment, we test the average number of maximum insertions and deletions within a limited time (500ms) with large numbers of attributes (9, 10, 11, 12). Table 4.15 and Figure 4.15 show the results with graphs of different densities, using 9 attributes.

Table 4.15 Average number of insertions with different densities (9 attributes) in 500ms

	size	# of preferences	initial graph (# of edges)	reduced graph (# of edges)	Proposed	King	Reconstruction
# of Inserts	9	512	510	498	1,206.70	31.49	140.95
# of Inserts	9	5,120	3,294	740	747.80	32.57	116.27
# of Inserts	9	18,944	10,253	680	1,872.85	35.26	97.80
# of Inserts	9	41,984	21,671	643	3,051.50	37.26	80.73
# of Inserts	9	74,240	38,008	642	2,741.52	38.76	64.75
# of Inserts	9	115,712	58,326	620	3,765.25	39.82	52.34
# of Inserts	9	166,400	83,211	566	16,736.24	39.74	43.38
# of Inserts	9	226,304	112,525	541	21,770.48	39.52	37.13

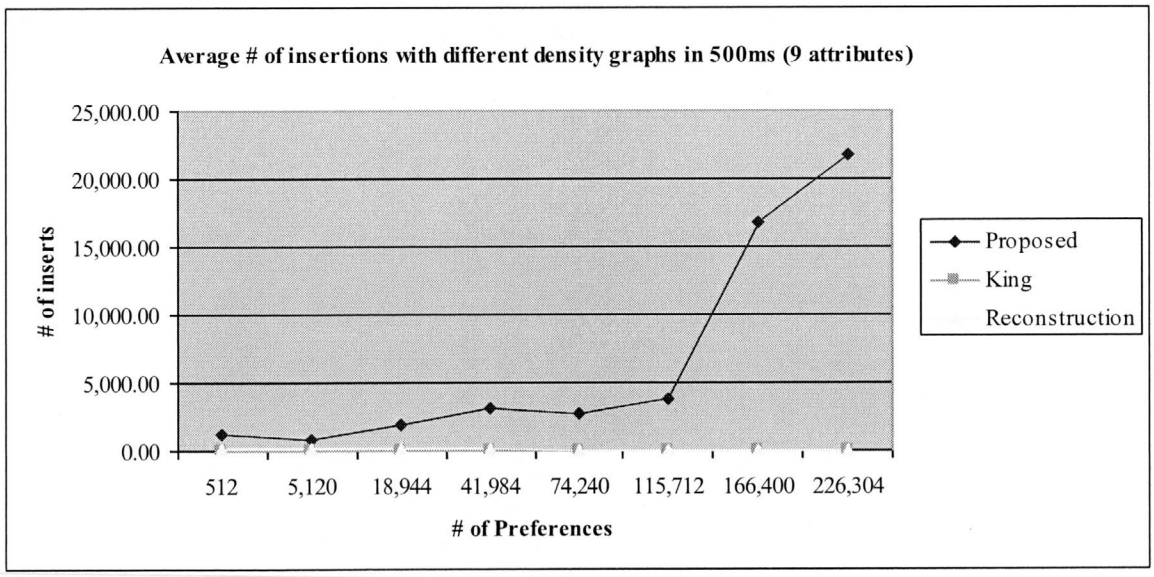


Figure 4.15 Average number of insertions with different density graphs (9 attributes)

Since the initial construction time takes a long time for large numbers of attributes, we only test *sparse* graphs for numbers of attributes greater than 9. Table 4.16 and Figure

4.16 show the average number of insertions in sparse graphs in a given time (500ms) using different numbers of attributes. Figure 4.17 shows two different ratios: 1) a ratio of the number of insertions with the proposed method to the number of insertions with King's method, 2) a ratio of the number of insertions with the proposed method to the number of reconstructions. In these results, we can see the proposed method is the most efficient among three different approaches, for large numbers of attributes.

Table 4.16 Average number of insertions in sparse graphs in 500ms

	size	# of preferences	initial graph (# of edges)	reduced graph (# of edges)	Proposed	King	Reconstruction
# of inserts	9	512	508.00	501.00	1,191.38	31.03	138.42
# of inserts	10	1,024	1,019.00	1,013.00	601.22	8.07	36.21
# of inserts	11	2,048	2,044.00	2,028.00	293.68	1.99	9.89
# of inserts	12	4,096	4,092.00	4,074.00	129.98	1.00	2.97

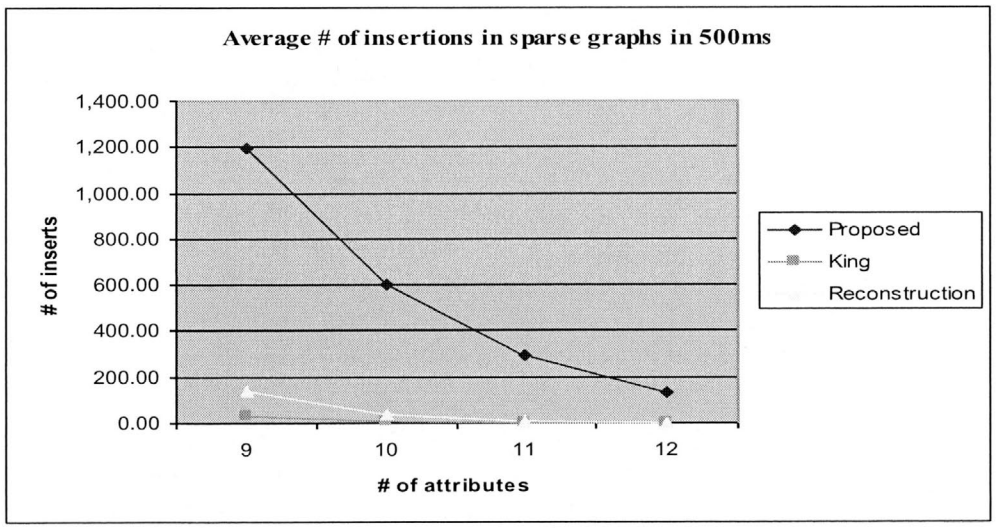


Figure 4.16 Average number of insertions in sparse graphs in 500ms

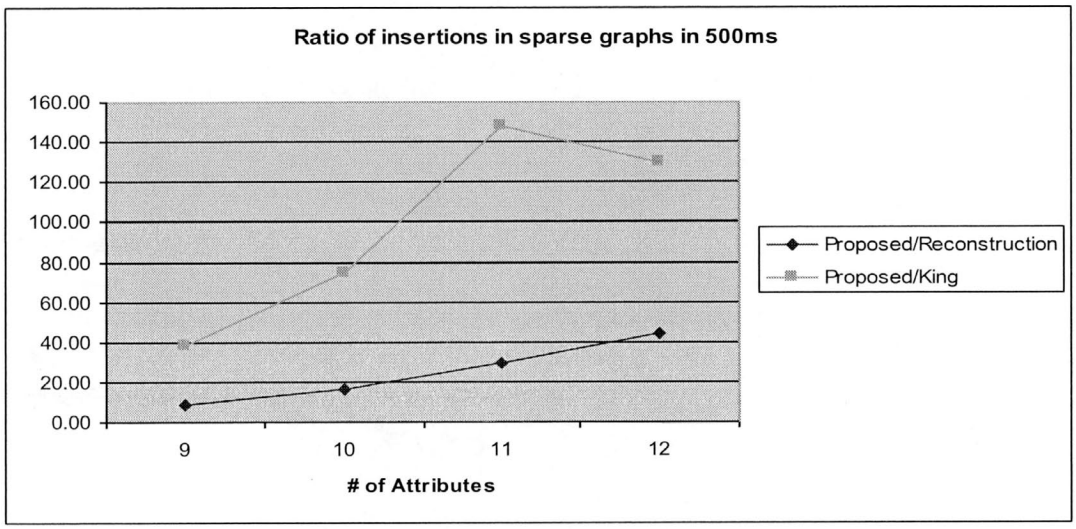


Figure 4.17 Ratio of insertions in sparse graphs in 500ms

Table 4.17 and Figure 4.18 show the average maximum number of deletions with different density graphs using 9 attributes. Table 4.18 and Figure 4.19 show the average maximum number of deletions with *sparse* graphs, using different numbers of attributes.

Table 4.17 Average number of deletions with different densities (9 attributes) in 500ms

	size	# of preferences	initial graph (# of edges)	reduced graph (# of edges)	Proposed	King	Reconstruction
# of Deletes	9	512	507	501	161.22	10.94	140.69
# of Deletes	9	5,120	3,223	732	23.43	20.16	116.81
# of Deletes	9	18,944	10,370	675	21.08	22.91	97.70
# of Deletes	9	41,984	22,008	667	21.91	24.59	80.68
# of Deletes	9	74,240	37,932	637	19.13	24.17	64.38
# of Deletes	9	115,712	58,288	602	17.74	25.38	52.61
# of Deletes	9	166,400	83,396	570	15.41	25.99	43.78
# of Deletes	9	226,304	112,873	540	14.08	26.92	37.33

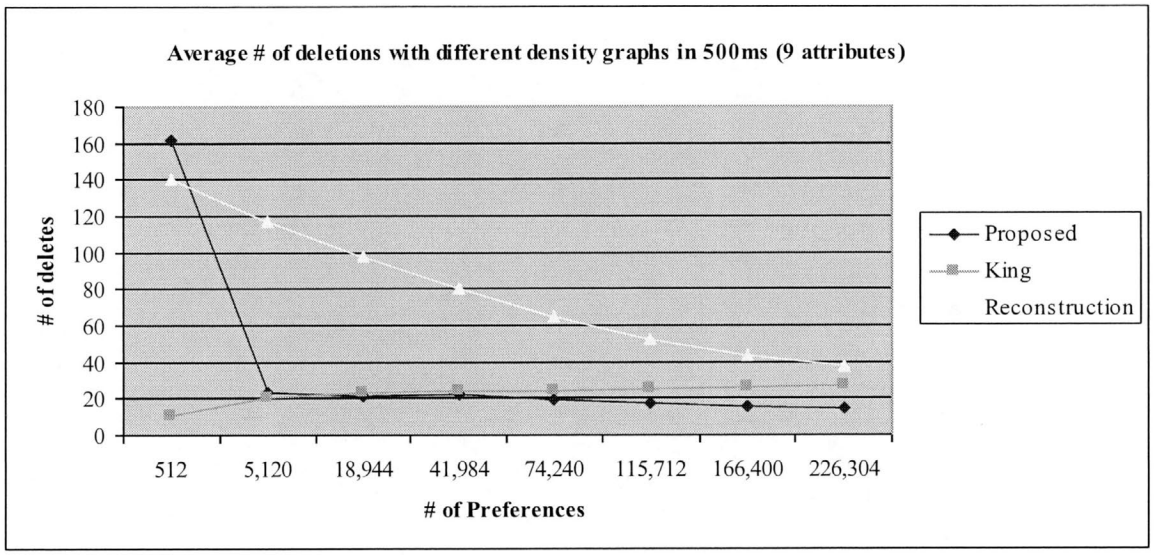


Figure 4.18 Average number of deletions in different density graphs (9 attributes)

Table 4.18 Average number of deletions in sparse graphs in 500ms

	size	# of preferences	initial graph (# of edges)	reduced graph (# of edges)	Proposed	King	Reconstruction
# of deletes	9	512	508.00	504.00	192.63	10.92	140.56
# of deletes	10	1,024	1,022.00	1,015.00	75.14	2.97	36.43
# of deletes	11	2,048	2,044.00	2,030.00	28.85	0.99	9.88
# of deletes	12	4,096	4,090.00	4,075.00	12.22	0.99	2.97

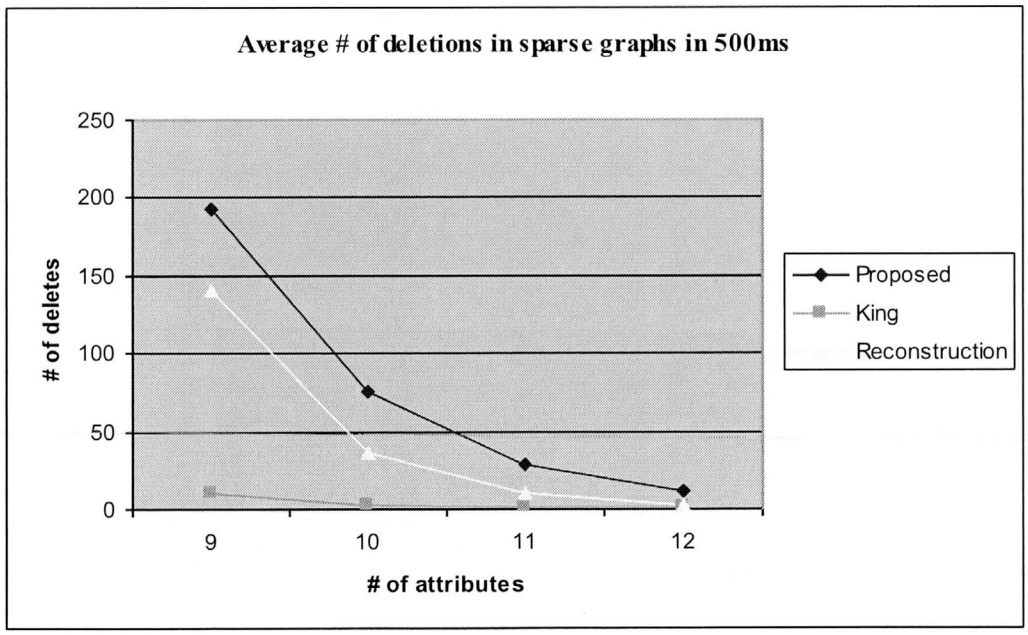


Figure 4.19 Average number of deletions with sparse graphs in 500ms

Figure 4.20 shows two different ratios: 1) a ratio of the number of deletions with the proposed method to the number of deletions with King's method, 2) a ratio of the number of deletions with the proposed method to the number of reconstructions. Each average maximum number of deletions shows that the reconstruction is more efficient than maintaining a reduced COP-network using either King's deletion algorithm or the proposed deletion algorithm, regardless of the size of graph, except in the case of *sparse* graphs. Since the reconstruction method still needs to check its consistency in order to find the longest path to estimate utilities, the proposed method is more efficient with a large number of attributes, for sparse graphs. For dense graphs, King's method is always more efficient than the proposed method.

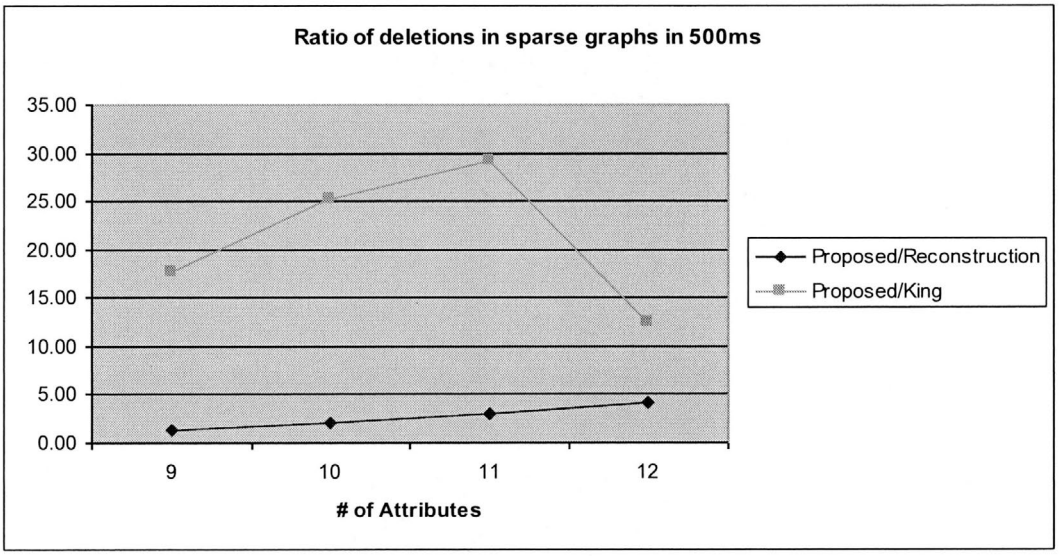


Figure 4.20 Ratio of deletions in sparse graphs in 500ms

Finally, we can notice that the proposed method is the most efficient with building initial graphs and with inserting new preferences regardless of the number of attributes and the graph's density. However, the proposed method is less efficient in deleting preferences with a small number of attributes and with denser graphs.

4.2 Analysis

Of the three approaches considered, the proposed method achieves the best results for inserting new preferences. Since our new insertion algorithm only keeps track of the status between two vertices, once we reduce an edge, we do not need to update it later. Therefore, if a graph is denser, the insertion time of the proposed method is reduced more. However, because of keeping track of the status, our deletion algorithm takes more time to update the status after deleting a preference. In practice, changes in user preferences are more likely to involve adding new preferences than deleting preferences. Therefore, reducing insertion time is a more important part of the entire maintenance process. The proposed method boasts the smallest insertion time, compared to the current existing methods.

Also, our new proposed method handles larger numbers of attributes than the current existing method. Since the existing method is based on keeping the number of paths between two vertices, it requires an object matrix, whereas the proposed method uses two bit matrices. As a result, the proposed method reduces its space and its running time.

5. Conclusions and Future work

Finally, in this chapter, we will provide a brief summary of the thesis, thesis contributions and future work.

5.1 Summary

Since preferences have been considered as an important part of automated decision making, there has been a great deal of research done on preference elicitation models. Most user preference elicitation models have been developed based on the assumption that user preferences are stable. However, user preferences may change in the long term and may evolve with experience, resulting in dynamic preferences. Therefore, most current user preference elicitation models are faced with rebuilding the entire preference model when a user's previous preferences are changed. Therefore, in this thesis, we consider maintaining a model that can adapt to changing user preferences.

We developed a dynamic user preference elicitation model that does not require the reconstruction of an entire preference model when user preferences are changed. In order to do this, we improved upon Chen et al.'s reduced COP-network model[2]. A COP-network is an acyclic graph that represents conditional user preference relations. A reduced COP-network is a transitive reduction of an initial COP-network that makes it easy to estimate utilities using a technique called the longest path method. In order to maintain a COP-network with changing preferences and to find reduced COP-networks easily, we improved upon King's algorithm[3], which has $O(n^2)$ computational time to maintain a transitive reduction. Since King's algorithm is based on a path matrix that

keeps track of the number of distinct directed paths between two vertices, it needs to update each affected matrix entry after any edge, which represents a preference, is inserted or deleted. We first defined 4 basic conditions that can occur between two vertices i and j.

- Condition 1: there is no *path* and no edge from i to j.
- Condition 2: there is an edge (i, j) but no other directed path.
- Condition 3: there is no edge but a directed path from i to j.
- Condition 4: there is an edge (i, j) and a longer directed path from i to j.

In order to represent these 4 conditions, we maintain two $n \times n$ binary matrices, where n is the number of vertices in a COP-network. Matrix 1 stores whether or not there is an edge between each pair of vertices, and Matrix 2 stores whether or not there is a directed path. Therefore, Condition 1 can be represented with Matrix 1 and Matrix 2 having values of 0. Condition 2 can be represented with Matrix 1 having a value of 1 and Matrix 2 having a value of 0. Condition 3 can be represented by Matrix 1 having a value of 0 and Matrix 2 having a value of 1. Condition 4 can be represented by both Matrix 1 and Matrix 2 having values of 1. Based on these 4 conditions, we developed new insert and delete algorithms.

We showed that our new algorithms are more efficient than King's algorithms except in the case of deletion when there is a small number of attributes or when the graph is particularly dense. Our insertion algorithm shows the best results with large numbers of attributes and with dense graphs. We showed that we could reduce the number of updates, compared to King's algorithm, since there is no need to keep the exact number of directed paths between any pair of vertices.

In our evaluation of our algorithms and King's, we compared the average running time to build an initial preference network and the update time for inserting/deleting a single new preference. We also compared the maximum number of updates (insertions/deletions) that could be performed within a given time. Our evaluation results clearly showed that the proposed method is efficient in maintaining a model of preferences in a dynamic COP-network. Although it sacrifices some efficiency in deleting preferences from denser graphs, it shows significant savings in insertion time and in space. Since, in practice, deleting preferences is caused mostly by direct conflicts after adding new preferences, reducing insertion time can lead to overall improvement.

5.2 Thesis Contributions

In this thesis, we provide a dynamic preference model that can support new preference addition as well as preference deletion after an initial preference model has been built. Our main contribution is that we apply the concept of transitive closure maintenance to the problem of maintaining an evolving preference model, which includes the specification of new insertion and deletion algorithms. The second contribution is that the space and average time needed to perform this maintenance is significantly reduced in many cases by utilizing a binary representation of the dynamic preference model.

5.3 Future Work

Since preference elicitation models have been considered as an important part of automatic decision making, our dynamic COP-network can be used for more efficient and more flexible preference elicitation to support real-time decision making. For future work, we could investigate whether running time and space could be improved further if we were to decompose a preference graph into smaller graphs based on attribute independence. We could find efficient preference utility elicitation functions for each portion of the preference model after decomposing a large user preference model into a number of smaller graphs. Another direction for future work is to investigate more practical applications, such as personalized search engines or personalized customer relational management systems. Our proposed method can help in generating dynamic personal preference models and in maintaining changing preferences.

Bibliography

[1] S. Buffett and M. Fleming, "Persistently Effective Query Selection in Preference Elicitation" in The 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07), pp. 491-497, Fremont, California, USA. 2007

[2] S. Chen, S. Buffett, and M. Fleming, "Reasoning with Conditional Preferences Across Attributes" in 20th Canadian Conference on Artificial Intelligence, pp. 369-380, 2007

[3] V. King and G. Sagert, "A Fully Dynamic Algorithm for Maintaining the Transitive Closure", Journal of Computer and System Sciences 65, pp.150–167, 2002

[4] AV. Aho, M.R. Garey and J.D. Ullman, "The Transitive Reduction for a Directed Graph", SIAM J. Computing, 1(2), pp.131-137, June 1972

[5] F. Miguel, M., M. Ryan, and A. Scott, "Are preferences stable? The case of health care" in Journal of Economic Behavior & Organization, Vol. 48, pp. 1~14, 2002

[6] T. Koopmans, "On flexibility of future preference" in Cowles Foundation Discussion Papers, 1962

[7] C. Boutilier, R. Brafman, H. Hoos, and D. Poole, "Reasoning with conditional ceteris paribus preference statements" in Proceedings of Uncertainty in Artificial Intelligence, 1999

[8] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole, "CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements" in Journal of Artificial Intelligence Research 21, pp135-191, 2004

[9] P.Chang and L.J. Henschen, "Parallel Transitive Closure and Transitive Reduction Algorithms", Databases, Parallel Architectures and Their Applications, ARBASE-90, pp.152-154, 1990

[10] M. McGeachie, "Recompiling Utility Functions in a Changing World" in AAAI Spring Symposium on Challenges to Decision Support in a Changing World, pp. 80-85, 2005

[11] J.A. La Poutré and Jan Van Leeuwen, "Maintenance of transitive closures and transitive reductions of graphs", 1988

[12] Henry Bediako-Asare, "Using A* Search to Generate Partial Preference networks", MCS thesis at University of New Brunswick, 2010

[13] P. Pu, B. Faltings, and M. Torrens, "Effective interaction principles for on-line product search environments" in Proceedings of the 3rd ACM/IEEE International Conference on Web intelligence. IEEE Press, 2004

[14] J. Zhang, P. Pu, and B. Faltings, "Agile decision agent for service-oriented e-commerce systems" in Proceedings of The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'06), pp. 182-189, 2006.

[15] V. Ha and P. Haddawy, "Problem-focused incremental elicitation of multi-attribute utility models", in Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, pp. 215-222, 1997

[16] M. L. Case, "Online Algorithms To Maintain A Transitive Reduction," [Online], Available: http://case-home.com/publications/caseCS2948.pdf, Accessed 24 Nov. 2011

[17] M. McGeachie and J. Doyle, "UTILITY FUNCTIONS FOR CETERIS PARIBUS PREFERENCES," Computational Intelligence, Vol. 20, Issue 2, pp. 158–217, 2004

[18] S. O. Hansson, "What is ceteris paribus preference?" Journal of Philosophical Logic, Vol. 25, No. 3, pp. 307-332, 1996

[19] R. L. Keeney, "Multiattribute utility analysis: a brief survey," International Institute For Applied Systems Analysis, Research Memorandum, 1975.

[20] R. L. Keeney and H. Raiffa, "Decisions with Multiple Objectives: Preferences and Value Tradeoffs," Wiley, New York, 1976.

[21] J. Doyle, Y. Shoham and M. P. Wellman, "A logic of relative desire (preliminary report)," Methodologies for Intelligent Systems, Lecture notes in Computer Science, Vol. 542, pp.16-31, 1991

Curriculum Vitae

Candidate's full name: Ki Hyang Lee

Universities attended (with dates and degrees obtained):

May 2009, Bachelor of Computer Science

University of New Brunswick

Feb. 1996, Bachelor of Computer Engineering

Hanbat National University, Republic of Korea

Presentations:

• Ki Hyang Lee, Michael Fleming and Scott Buffett, "Maintaining Dynamic COP-nets with Changing Preferences" in Proceeding of the 8th annual research exposition, p. 17, UNB, Fredericton, 2011.