HEURISTICS FOR AUTOMATED KNOWLEDGE SOURCE INTEGRATION AND SERVICE COMPOSITION

Patrick N. Bless
Department of Mechanical & Industrial Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois
Email: pbless@gmail.com

Diego Klabjan
Department of Mechanical & Industrial Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois
Email: klabjan@uiuc.edu

Soo Y. Chang
Department of Industrial Engineering
Pohang University of Science and Technology - Hyoja San 31,
Pohang, Republic of Korea
Email: syc@postech.ac.kr

ABSTRACT

The *NP*-hard component set identification problem is a combinatorial problem arising in the context of knowledge discovery, information integration, and knowledge source/service composition. Considering a granular knowledge domain consisting of a large number of individual bits and pieces of domain knowledge (properties) and a large number of knowledge sources and services that provide mappings between sets of properties, the objective of the component set identification problem is to select a minimum cost combination of knowledge sources that can provide a joint mapping from a given set of initially available properties (initial knowledge) to a set of initially unknown properties (target knowledge). We provide a general framework for heuristics and consider construction heuristics that are followed by local improvement heuristics. Computational results are reported on randomly generated problem instances.

1 Introduction

In many areas, such as natural science, engineering, and medicine researchers are accumulating an ever-growing amount of domain knowledge. Even though this is not a new phenomenon, the methodologies and tools to generate, store, and utilize domain knowledge have been changed fundamentally by the digital revolution. Disciplines such as data mining, knowledge discovery, machine learning and artificial intelligence are just a few examples of efforts that have been undertaken to successfully cope with and take full advantage of an ever-increasing amount of knowledge and information.

While the methodologies and tools of knowledge management are continuously evolving, the fundamental concepts of accessing knowledge and retrieving information remain unchanged. No matter whether information is being retrieved or requested from a living organism, a printed log table, a finite element model, a neural network, a web service, or a sensor attached to a satellite, the basic transaction required to obtain information or knowledge from any of these sources can be modeled as a function that converts a set of input arguments (provided information) into a set of output arguments (sought information). Any such function can be defined as a knowledge source. This very broad and rather loose definition of the notion of a knowledge source shows the wide reaching applicability and relevance of the presented combinatorial problems.

Digital knowledge sources have become a dominant concept for distributed computing and the use of communication networks as an aggregation point for heterogeneous knowledge sources and services is becoming more and more important. One example for such digital knowledge sources are web services, which can be described as applications that can be published, located, and accessed from anywhere in the network, McIlraith (2001). Web services are fully characterized by the messages they send and receive, and by the operations or services they provide, Christensen *et al.* (2001). The aforementioned definition of a knowledge source includes the definition of web services. A substantial amount of work has been done on the description and the syntax of web services, Staab (2003),

and the declared future plan is to develop methodologies for automatic web services composition. Different approaches have been proposed to pursuit this objective, including web service choreography, Arkin *et al.* (2002), and web service orchestration, Peltz (2003). Web service choreography tracks the sequence of messages among multiple parties and multiple sources, and web service orchestration deals with the issue of how web services can interact and communicate with each other at the message level, including the business logic and execution order of the interactions.

Besides these relatively new and very specific efforts, an abundant amount of research has focused on the overarching problem of logical and physical integration of heterogeneous information. A large number of different approaches (integration paradigms) have been considered, including, e.g., the concepts of object orientated programming, reusable software modules, Cheng and Jeng (1997), ontology engineering, Chandrasekaran *et al.* (1999), and middleware standards, Bakken (2001).

The component set identification (CSI) problem can be described as the problem of identifying a combination of knowledge sources that can provide a joint mapping from available information to certain pieces of sought information or knowledge. Assuming that there are costs associated with the consultation of each knowledge source, the problem can be stated as a combinatorial optimization problem. It is shown by Bless *et al.* (2006) that the CSI problem is strongly *NP*-hard. They also give integer programming formulations for solving the problem, which show that large instances need to be solved by other techniques. Research efforts should therefore be focused on the development of efficient heuristics that are able to (1) find near optimal solutions for small and medium size problem instances quicker and more reliably than exact solution procedures, and (2) find good feasible solutions to problem instances that are too large to be even approximated with exact solution procedures.

Our objective is to provide a comprehensive discussion of the design of heuristics for the CSI problem and to gain a fundamental understanding of the difficulties that arise when solving large instances of this new combinatorial problem. To facilitate the present and also any future development of heuristics for the CSI problem, we develop a general framework for the design of construction and local improvement heuristics. Based on this framework, nine unique classes of heuristics are developed, including four construction and five local improvement heuristics. All heuristics are supported by detailed computational results. Since the CSI problem has only recently been introduced, all of the above are also significant contributions of our work.

Bless *et al.* (2006) and this manuscript study the CSI problem from a complimentary point of view. While the former study is focused on formally defining and categorizing the problem, and on integer programming aspects of the problem together with an exact solution methodology for small to medium size problems, the current document exploits heuristics for solving the problem and performs a computational study for large-scale instances.

The remainder of this paper is structured as follows. In Section 2 we develop a general framework for the design of construction heuristics for the CSI problem. Based on this framework four different heuristics are developed, including a basic search heuristic, two greedy-type heuristics, and a shortest path heuristic. For each heuristic we present several variants. Section 3 first provides a general framework for the design of local improvement heuristics. Using this framework five local improvement heuristics with different neighborhood functions are presented. Section 4 provides a summary of results obtained from a variety of computational experiments. Within this section, we first provide a comparison of the individual performance of the different construction heuristics. Next, we present results for a similar comparison of the five different local improvement heuristics. In the last part of this section, the performance of all construction and local improvement heuristics is compared against the performance of an exact methodology. Finally, computational results on large problem instances are provided. We conclude the introduction with a formal problem statement and the literature review.

Formal Problem Statement for the Component Set Identification Problem

Any knowledge domain can be represented by a finite set $\mathbf{P} = \{p_j : j = 1,2...,m\}$ of knowledge bits or properties and a set of services or knowledge sources $\mathbf{K} = \{k_i : i = 1,2,...n\}$. The properties can be thought of as a granular representation of all relevant domain knowledge and the set of knowledge sources represents all available mappings from certain sets of required input properties to certain sets of output properties. Knowledge source k_i maps input properties $\mathbf{P}_i^{in} \subseteq \mathbf{P}$ to output properties $\mathbf{P}_i^{out} \subseteq \mathbf{P}$. Assuming a domain representation with a complete set of property and knowledge source libraries, every information or knowledge request that can potentially be made in this domain, can be characterized by two sets; a set $\mathbf{P}^I \subseteq \mathbf{P}$ of known properties (available knowledge) and a set $\mathbf{P}^T \subseteq \mathbf{P}$ of target properties (sought knowledge). Let $\tilde{p}^I = |\mathbf{P}^I|$ and $\tilde{p}^T = |\mathbf{P}^T|$. Assuming that each knowledge source has a non-negative cost associated with it, the objective of the CSI problem is to find the *minimum cost* combination of knowledge sources that can provide a joint mapping from $\mathbf{P}^I$ to $\mathbf{P}^T$. Depending on the knowledge source costs, and depending on whether a single cost criteria or a weighted combinations of multiple cost criteria is used, minimum cost can correspond to fastest, most accurate, most affordable combination, or a combination of these criteria.

The underlying decision problem can be modeled as a directed network consisting of a set $\mathbf{P}$ of m property nodes, a set $\mathbf{K}$ of n knowledge source nodes, and a set of arcs $\mathbf{A}$ connecting property nodes with knowledge source nodes. More specifically, there exists a directed arc (p_j, k_i) in the network if and only if property p_j is an input property of knowledge source k_i ($p_j \in \mathbf{P}_i^{in}$). Furthermore,

there exist a directed arc (k_i, p_j) if and only if property p_j is an output property of knowledge source k_i ($p_j \in \mathbf{P}_i^{out}$). The CSI problem is formally stated as follows. Given the network $G(\mathbf{P}, \mathbf{K}, \mathbf{A}, c)$, consisting of the described property and knowledge source sets, a set of arcs $\mathbf{A}$, a set of root nodes $\mathbf{P}^I$, a set of terminals $\mathbf{P}^T$, and nonnegative knowledge source cost function c, the objective is to find the minimum cost acyclic subnetwork rooted at $\mathbf{P}^I$ that spans all target nodes $\mathbf{P}^T$ and meets all *input-property-induced precedence constraints*. A knowledge source node $k_i \in \mathbf{K}$ can only be part of a feasible subnetwork if all property nodes $p_j \in \mathbf{P}_i^{in}$ are connected with k_i via a directed path in the subnetwork starting at one of the root nodes. This is equivalent to requiring that

- if k_i is in the subnetwork, then all p_j, $j \in \mathbf{P}_i^{in}$ are in the subnetwork as well (if a knowledge source is used, all of its input properties have to be known), and
- if $p_j \in \mathbf{P} \backslash \mathbf{P}^I$ is in the subnetwork, then at least one k_i, $j \in \mathbf{P}_i^{out}$ is in the subnetwork (property has to be returned by at least one knowledge source), and
- all $p_j \in \mathbf{P}^T$ are in the subnetwork.

The input size of the CSI problem is determined by the number of property nodes m, the number of knowledge source nodes n, the number of target properties $\tilde{p}^T$, the number of initially known properties $\tilde{p}^I$, and the knowledge source cost function c.

Finding the minimum cost acyclic subnetwork corresponds to finding the combination of knowledge sources that can generate the sought target properties $\mathbf{P}^T$ in the *most desirable* fashion. Figure 1 shows an example problem with three initially known properties (root nodes) and two target properties. The feasible solution marked in bold, provides paths from the initially known properties (P_B^I, P_C^I, P_F^I) to both target properties (P_x^T and P_y^T). The subnetwork consists of six (*active*) knowledge sources (K_C, K_G, K_H, K_K, K_L, and K_Z) and six intermediate property nodes (P_G, P_Q, P_L, P_M, P_R, and P_S). A knowledge sources is considered active if it is used within the solution subnetwork. The example shows both, knowledge sources that require a single input property and knowledge sources that require multiple input properties. Similarly, there are knowledge sources with a single output and knowledge sources with multiple outputs. All input-property-induced precedence constraints are satisfied.

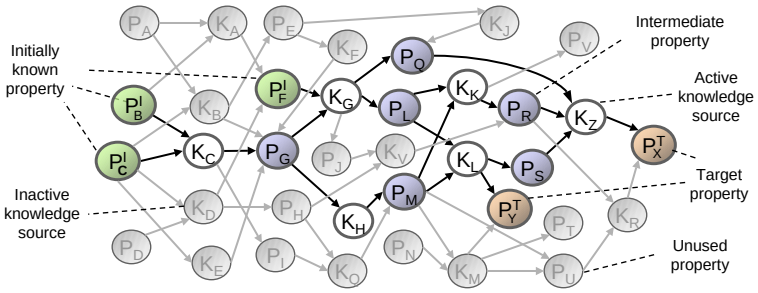

Figure 1. A CSI EXAMPLE.

We stress that a feasible solution to a CSI problem instance has to be 'implementable', i.e., the set of selected knowledge sources must satisfy the input-property-induced precedence constraints. Consider the example shown in Figure 2. At first the depicted solution seems to be feasible since there exists a path from the initially known properties, P_1 and P_2, to the single target property P_9. However, knowledge source K_2 generates property P_5, a required input for knowledge source K_3, and knowledge source K_3 generates property P_4, a required input property for knowledge source K_2. This clearly results in a deadlock, causing the solution to be infeasible. This solution clearly has a cycle. By definition, a knowledge source can only be used in a solution if all its required input properties are available. The availability of these input properties is a function of the set of knowledge sources that have been used prior to a knowledge source. The scheduling aspect is discussed in more detail in Bless *et al.* (2006).

The in- and out-degree p_i^{in}, p_i^{out} of knowledge source i is $|\mathbf{P}_i^{in}|$, $|\mathbf{P}_i^{out}|$, respectively. It corresponds to the in- and out-degree of k_i in $G(\mathbf{P}, \mathbf{K}, \mathbf{A}, c)$. These two CSI network characteristics are usually not constant for all knowledge sources. Similarly, the in- and out-degree of a property corresponds to the in- and out-degree of the corresponding node in $G(\mathbf{P}, \mathbf{K}, \mathbf{A}, c)$.

To simplify the discussion of the complexity of the various presented heuristics, let $N = min\{n, m\}$.

Literature Review

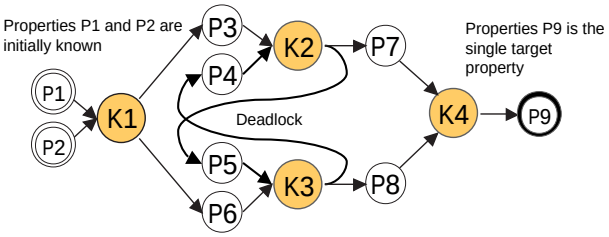

Figure 2. A DEADLOCK EXAMPLE.

The fundamental difficulty with the CSI problem arises from two different aspects of its combinatorial nature that are closely intertwined. In order to obtain a feasible solution to CSI, we must select the *right* set of knowledge sources and, at the same time, arrange them to form a subnetwork that satisfies the input-property-induced precedence constraints.

It is shown in Bless *et al.* (2006) that selecting the least cost set of knowledge sources is at least as difficult as the *vertex cover* problem, which is known to be *NP*-complete, Garey and Johnson (1979), Coppersmith and Vishkin (1985). As discussed in more detail in Bless *et al.* (2006), the arrangement of knowledge sources and their input/output properties in order to generate a feasible subnetwork has a combinatorial nature similar to the *Steiner network problem*, which is also known to be *NP*-complete, Garey and Johnson (1979), Hwang *et al.* (1992).

Vertex cover is just one of many different types of covering problems that are extensively treated in the optimization literature. In some applications, covering a graph with a subgraph with small maximum degree is sought, Lawrence (1978), or a subset of edges instead of nodes to cover only a specified subset of vertices is to be found, Plesnik (2001). Although an exact optimal algorithm can be developed for a special class of graphs, Fernau and Niedermeier (2001), most of the proposed algorithms are fast approximation heuristics developed for specific classes of graphs, Meek and Parker (1994), Halperin and Srinivasan (2002). Also, the weighted vertex cover problems are treated in Bar-Yehuda and Even (1981) and Niedermeier and Rossmanith (2003).

The Steiner tree problem is perhaps the most explored form of the Steiner network problem, which is studied often within the context of VLSI design, Groetschel *et al.* (1997). The Steiner network problem is generalized and various variants are derived in Dror and Haouari (2000) and it is studied under additional constraints by Rosenwein and Wong (1995) and Voss (1999). An optimal algorithm based on branch and cut is proposed in Lucena and Beasley (1998), but most of the algorithms are heuristics based on local search, Verhoeven (1999), Zachariasen (1999). The Steiner network problems on directed graphs are treated in Maculan *et al.* (1991) and the cases with multiple terminal nodes and node weights are handled in Fuchs (2003) and Engevall *et al.* (1998), respectively. Nevertheless, to the best of our knowledge, there is no algorithm that can be directly applied to CSI. Hence, we develop a class of efficient heuristics for CSI and derive more sophisticated variants of them by utilizing relatively simple forms of network models like shortest path.

General treatment of heuristics can be found in Aarts and Lenstra (1997) and Glover and Kochenberger (2002). A class of genetic local search techniques for various combinatorial optimization problems is discussed in Kolen and Pesch (1994). A comprehensive survey on large-scale local search heuristics can be found in Ahuja *et al.* (2002). Extensive computational experiments on neighborhood search techniques can be found in Steinhöfel *et al.* (2003). Heuristics for vertex cover are found in Bar-Yehuda and Even (1981), Halperin and Srinivasan (2002), and Meek and Parker (1994), whereas heuristics for the Steiner network problem are treated in Verhoeven (1999) and Zachariasen (1999).

2 Construction Heuristics

Construction heuristics for the CSI problem successively enlarge the set of selected knowledge sources until all target properties and all input properties of the selected knowledge sources are covered. Construction heuristics do not attempt to improve the obtained solution. Hence, knowledge sources that have already been selected and portions of the solution that have already been built, remain unchanged during the course of the solution procedure. We first present a general framework that facilitates the development of construction heuristics for the CSI problem. Based on this framework four specific construction heuristics are developed, including a basic search heuristic, two greedy-type heuristics, and a shortest path heuristic. For each heuristic we present several variations.

2.1 General Framework for Construction Heuristics

The development of all construction heuristics presented in this paper is based on the general framework depicted in Figure 3. Let w be a counter indicating the current iteration index, and let $\mathbf{P}_w^{CT}$ denote the set of target properties at the beginning of iteration w. $\mathbf{K}_w^S$ is the set of all knowledge sources selected prior to iteration w and $\mathbf{K}_w^{SI}$ is the set of knowledge sources that are selected in iteration w. Finally, $\mathbf{P}_w^{CA}$ is defined as the set of properties that are available (known) at the beginning of iteration w.

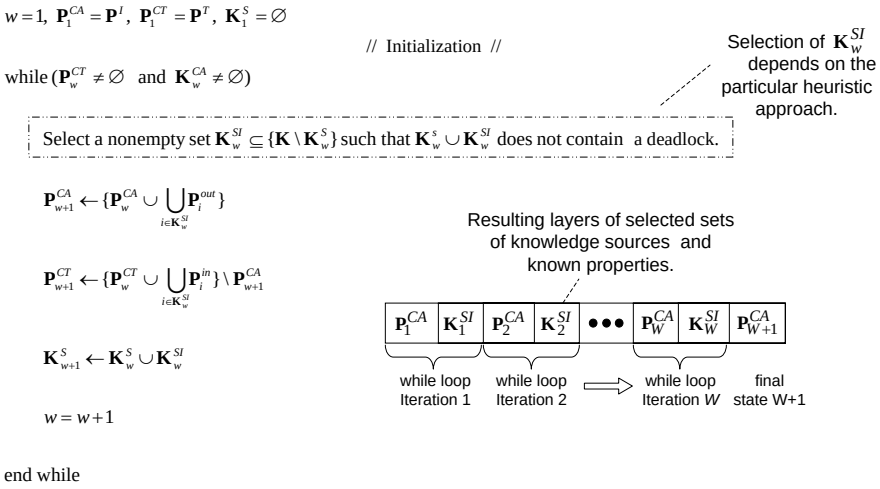

Figure 3. FRAMEWORK FOR CONSTRUCTION HEURISTICS.

For all construction heuristics presented in this paper, $\mathbf{P}_1^{CT}$ is initialized with the set of target properties $\mathbf{P}^T$, $\mathbf{P}_1^{CA}$ is initialized with the set of initially known properties $\mathbf{P}^I$, and $\mathbf{K}_1^S$ is initialized with the empty set $\varnothing$. In each iteration of the while loop the heuristic selects a set $\mathbf{K}_w^{SI}\subseteq\mathbf{K}\setminus\mathbf{K}_w^S$ of knowledge sources to be added to the set of previously selected knowledge sources. The selection criterion varies for the different construction heuristics. The discussion of these criteria is provided for each construction heuristic individually in the sections that follow. After the set $\mathbf{K}_w^{SI}$ has been determined, sets $\mathbf{P}_w^{CA}$, $\mathbf{P}_w^{CT}$, and $\mathbf{K}_w^S$ are updated for the next iteration. More specifically, $\mathbf{P}_w^{CA}$ is updated to $\mathbf{P}_{w+1}^{CA}$ by adding all properties that are returned by the newly selected knowledge sources. $\mathbf{P}_w^{CT}$ is updated to $\mathbf{P}_{w+1}^{CT}$ by first adding all required input properties of the newly selected knowledge sources and then removing all properties in $\mathbf{P}_{w+1}^{CA}$. The selection process is repeated until one of two termination criteria, $\mathbf{P}_w^{CT}=\varnothing$ and $\mathbf{K}_w^S=\mathbf{K}$, is met. If the first termination criterion is met, the heuristic finds a feasible solution. Such a set is referred to as a *feasible set*. If the latter criterion causes the heuristic to terminate, the problem is infeasible.

The number of knowledge sources added to the set of selected knowledge sources in each iteration can vary from iteration to iteration and also depends on the particular construction heuristic that is being used. However, it is obvious that by definition of the general framework, at least one knowledge source is added to the set of selected knowledge sources in each iteration. Furthermore, it is clear that any reasonable heuristic will generate at least one previously unknown property in each iteration. Based on this observation it follows that a problem with n knowledge sources and m properties requires a maximum of N iterations. Given this general framework, the discussion of construction heuristics is reduced to the description of the selection criteria used to determine the set $\mathbf{K}_w^{SI}$ in each iteration.

In order to identify combinations of knowledge sources that result in 'good' solutions, the selection criteria of the majority of construction heuristics are based on specially designed scores for the knowledge sources. These do not necessarily have to be equal to the knowledge source cost labels c_i. Instead of using the plain cost information provided by the problem definition, there is a wide range of alternative choices for more intelligent score definitions. These alternative scores take advantage of additional information about the problem instance and about the current progress or state of the solution process. For example, intuitively selecting a knowledge source that provides many target properties is beneficial. Scores that vary from one iteration to the next are referred to as *dynamic scores*. Throughout this paper different choices for constant and dynamic scores are considered and differences in performance are discussed in detail. All scores can be described as functions of the available information about the problem instance and about the current state of the

partially constructed solution. A very general definition, covering all score definitions used in this study is

$$CK_i^H = f(c_i, \mathbf{P}_i^{in}, \mathbf{P}_i^{out}, \mathbf{P}_w^{CT}, \mathbf{P}_w^{CA}, \mathbf{K}_w^S),$$

where CK_i^H is the score of knowledge source i. The choice of f depends on the underlying heuristics.

2.2 The Firing Heuristic

The firing heuristic is designed to quickly determine whether a given problem instance has a feasible solution or not. In addition to quickly providing a feasible solution, the heuristic reveals important information about the problem instance, which is subsequently used in more advanced heuristics. The general selection criterion for the firing heuristic is

$$\mathbf{K}_w^{SI} = \{k_i | \mathbf{P}_i^{in} \subseteq \mathbf{P}_w^{CA}, k_i \notin \mathbf{K}_w^S\}. \tag{1}$$

According to (1), the heuristic selects every knowledge sources that can potentially be used. The algorithm scans the set of all knowledge sources that have not yet been selected and picks the ones for which all required input properties are available ($\mathbf{P}_i^{in} \subseteq \mathbf{P}_w^{CA}$) and therefore can be 'fired'. If at the end $\mathbf{K}_W^S = \mathbf{K}$ and $\mathbf{P}^{CT} \neq \varnothing$, then the problem is infeasible. Indeed, the CSI problem is feasible if and only if the firing heuristic finds a solution. The heuristic terminates when all target properties have been obtained or all knowledge sources have been selected. Since knowledge sources can only be selected if and only if all required input properties are available, the set of knowledge sources selected by the firing heuristic can never contain a deadlock. When implemented within the general framework, the time complexity of the heuristic is $O(n^2 p_{max}^{in} + n^2 p_{max}^{out}) = O(n^2 m)$, where p_{max}^{in} is the maximum in-degree, and p_{max}^{out} is the maximum out-degree of all knowledge sources in $\mathbf{K}$. Figure 4 illustrates the layered solution structure generated by the firing heuristic.

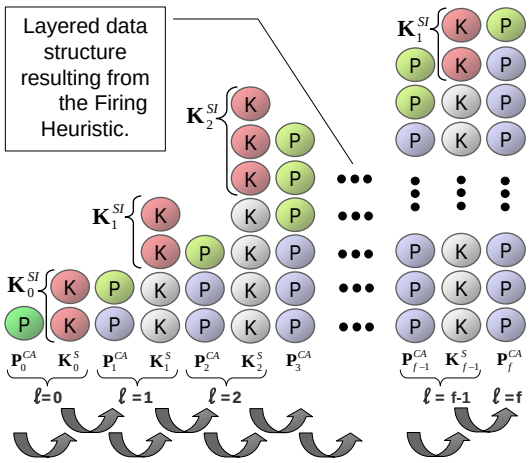

Figure 4. LAYERED DATA STRUCTURE OBTAINED FROM THE FIRING HEURISTIC.

Since the firing-heuristic ignores all cost considerations and simply selects the absolute maximum number of knowledge sources that can be fired, it minimizes the number of iterations required to obtain a feasible solution. As a results, the total number of layers in the solution represents a lower bound, F, for the number of layers in any feasible solution. The firing heuristic is the only heuristic for which the number of while-loop iterations of the general framework always exactly corresponds to the total number of layers L in the resulting solution.

The *length of a solution* is the minimum number of layers used by this solution. We emphasize that based on the above discussion, we can obtain the minimum length of a solution by running the firing heuristic on the set of selected knowledge sources, i.e., the firing heuristic can be used to obtain the minimum number of layers used by a solution no matter what type of solution method was used to obtain the solution.

Due to its brute force approach, the firing heuristic is prone to select many knowledge sources that are redundant or even completely unnecessary for the determination of the desired target properties. These redundant and unnecessary knowledge sources can be eliminated in a simple backwards sweep over the resulting layers of the solution. Even though the idea of a backward sweep is not in complete agreement with the definition of construction heuristics, which generally do not try to improve upon solutions, this *clean up* operation can be seen as an integral part of the firing heuristic. Figure 5 provides the pseudocode for the backward sweep operation.

$\ell = F$; $\mathbf{P}_F^{BCT} = \mathbf{P}^T$; $\mathbf{K}_F^{BS} = \emptyset$;

while ($\mathbf{P}_\ell^{BCT} \neq \emptyset$)

Remove from $\mathbf{K}^S$ all unnecessary knowledge sources from $\mathbf{K}_\ell^{SI}$ that do not provide a single target property

$$\mathbf{P}_{\ell-1}^{BCT} \leftarrow \mathbf{P}_\ell^{BCT} \cup \bigcup_{i \in \mathbf{K}_\ell^{SI}} \mathbf{P}_i^{in} \setminus \bigcup_{i \in \mathbf{K}_\ell^{SI}} \mathbf{P}_i^{out}$$

$\ell = \ell - 1$

end while

Figure 5. PSEUDO CODE FOR THE BACKWARD SWEEP OPERATION.

Let ℓ, $1 \leq \ell \leq F$ be the layer index. Starting at the last layer of the solution $\ell = F$, all knowledge sources in $\mathbf{K}_\ell^{SI}$ that do not provide at least one of the current target properties, i.e., are unnecessary, are removed from $\mathbf{K}^S$. In general the following set covering problem has to be solved. The ground set is the set of all current target properties that become known in the current layer, the subsets are the output property sets of all knowledge sources that are fired in the current layer ($k_i \in \mathbf{K}_\ell^{SI}$), and the cost of a subset is the cost of the corresponding knowledge source. We solve the set covering problem approximately with a greedy algorithm that picks low cost subsets. When no additional knowledge sources are removed from $\mathbf{K}_\ell^{SI}$, the new set of current target properties for layer $\ell - 1$ is set to $\mathbf{P}_{\ell-1}^{BCT} = \mathbf{P}_\ell^{BCT} \cup \bigcup_{i \in \mathbf{K}_\ell^{SI}} \mathbf{P}_i^{in} \setminus \bigcup_{i \in \mathbf{K}_\ell^{SI}} \mathbf{P}_i^{out}$ and the algorithm proceeds to layer $\ell = \ell - 1$. This backward sweep operation is repeated until $\mathbf{P}_\ell^{BCT}$ is empty. At the beginning of the heuristic, the set of current targets $\mathbf{P}_F^{BCT}$ is initialized with $\mathbf{P}^T$. The worst case complexity of the backward sweep operation is $O(Nn\log(n))$ and therefore the complexity of the overall heuristic $O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + Nn\log(n))$. (The term $n\log(n)$ comes from the sorting operation within the greedy set covering heuristic).

2.3 Backward Greedy Heuristic

The additional information generated by the firing heuristic can be used to develop an advanced class of construction heuristics, referred to as backward greedy (BG) heuristic. Besides the layered data structure and the lower bound on the overall length of any feasible solution, the firing heuristic provides a lower bound for the earliest layer in which a knowledge source and a property can be used or be generated. The first type of a lower bound, denoted by L_i^e, is associated with knowledge source i and provides the *earliest* layer in which knowledge source i can be used. This corresponds to the earliest layer in which all required input properties for knowledge source i can be made available. The second type of a lower bound, denoted by l_j^e, is associated with property j and provides the *earliest* layer in which property j can be made available. All of these values can be obtained by running the firing heuristic to the end, i.e., until all knowledge sources have been used or no more knowledge sources can be fired. In other words, the firing heuristic is not stopped when the first feasible solution has been found.

The BG-heuristic begins its search in layer $\ell = F$, which is the earliest layer in which all target properties can be obtained. Following the general framework, the heuristic works its way backwards through the layered data structure provided by the firing heuristic as shown in Figure 6. A heuristic selects knowledge sources and adds them to the solution subnetwork until all target properties are covered and the subnetwork meets all input-property-induced precedence constraints. In contrast to the firing heuristic, the BG-heuristic picks only one knowledge source per iteration ($|\mathbf{K}_w^{SI}| = 1$). The selection criterion used to determine $\mathbf{K}_w^{SI}$ in each iteration is based on the score CK_i^H, and on the lower bounds L_i^e ,$i \in \mathbf{K}$ for the earliest layer in which each knowledge source can be fired. It reads

$$\mathbf{K}_w^{SI} = \underset{i}{\operatorname{argmin}} \{CK_i^H | L_i^e \leq LMAX_w\},$$

where $LMAX_w = min\{L_q^e | q \in \mathbf{K}_w^S\}$. Initially $LMAX_1 = F$. Ties are broken arbitrarily.

Since the BG-heuristic is operating on the layered network provided by the firing heuristic, the heuristic can never generate a set of knowledge sources that contains deadlocking.

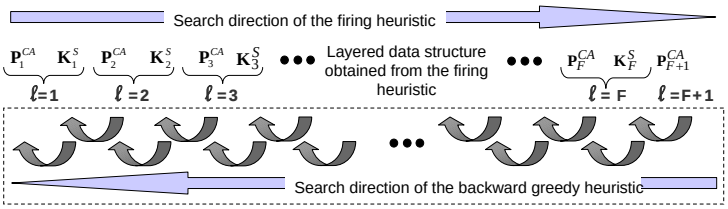

Figure 6. BASIC SEARCH DIRECTION AND RELATION BETWEEN THE BACKWARD GREEDY HEURISTIC AND THE FIRING HEURISTIC.

Several different versions of the BG-heuristic have been developed (see Table 1). Version A considers simple knowledge source scores based solely on the cost of knowledge sources, $CK_i^H = c_i$. Version B takes into account that some knowledge sources may provide more than one of the current target properties. Clearly, everything else being equal, we prefer a knowledge source that can generate many target properties. Hence, the cost of each knowledge source is divided by the cardinality of the intersection of the set of current target properties and the set of output properties of the knowledge source, $CK_i^H = c_i/|\mathbf{P}_w^{CT} \cap \mathbf{P}_i^{out}|$. Version C considers the net change $|(|\mathbf{P}_w^{CT}| - |\mathbf{P}_{w+1}^{CT}|)|$ in the size of the set of current target properties, resulting from the inclusion of a knowledge source. Therefore, the knowledge source score reads $CK_i^H = |\mathbf{P}_i^{in} \backslash \mathbf{P}_w^{CT}| - |\mathbf{P}_i^{out} \cap \mathbf{P}_w^{CT}|$. Versions D, E, F, and G try to further improve the performance by estimating the cost associated with obtaining all input properties of a knowledge source and incorporating these cost scores into the corresponding knowledge source score. These estimates are called property scores and can be obtained in a variety of different ways. Since property scores are used for the development of other heuristics as well, the definition of these quantities constitutes an important aspect of the development of heuristics for the CSI problem in general. Next we present the property scores used for the different versions of the BG-heuristic.

1. The first property score is based on the lower bound l_j^e, representing the earliest layer in which property j can be generated in any feasible solution. It is easy to see that in order to obtain p_j, at least l_j^e knowledge sources have to be used, one from each layer ℓ, $1 \leq \ell \leq l_j^e$. Assuming an average knowledge source cost of $\bar{c} = \sum_{i=1}^{n} c_i/n$, the score for property j becomes $CP_j^H = l_j^e \bar{c}$.
2. The second property score incorporates an additional characteristic of each property by considering the number of knowledge sources that provide property p_j as output. The rational behind this definition is that a property that has relatively few *sources* is more likely to be more difficult (expensive) to obtain than a property that is provided by a large number of knowledge sources. The resulting score of p_j reads $\overline{CP}_j^H = l_j^e \bar{c}/|\mathbf{S}_j|$, where $\mathbf{S}_j = \{k_i \in \mathbf{K} | j \in \mathbf{P}_i^{out}\}$ is the set of all knowledge sources that provide p_j as output.
3. The third property score is similar to the previous one. The difference is that now only those sources i of property j are considered that have a lower bound L_i^e that is equal or lower than the lowest L^e among all previously selected knowledge sources. The reasoning behind this choice is the following. Since the heuristic moves backwards, only knowledge sources that can be used between the current layer and the very first layer of the layered data structure obtained from the firing heuristic have to be considered. All other knowledge sources are clearly not available for use in layers 1 through the current one. Therefore, a modified set of sources, $\overline{\mathbf{S}}_j = \{k_i \in \mathbf{K} | j \in \mathbf{P}_i^{out}, L_i^e \leq min\{L_i^e | i \in \mathbf{K}_w^S\}\}$, is defined and the property score becomes $CP_j^H = l_j^e \bar{c}/|\overline{\mathbf{S}}_j|$.
4. The last type of a property score adds an additional level of complexity by introducing recursion into the design of construction heuristics for the CSI problem. Property scores are obtained by solving m modified CSI subproblems, one for each property. Each of the m subproblem instances is identical to the original one except that it has exactly one target property. The target property corresponds to the property for which a cost label estimate is sought. The total cost of the best solution for each subproblem is used as the score for the corresponding property. These subproblems are solved by version A of the BG-heuristic. We have $CP_j^H = E_j^H$, where E_j^H is the value of the solution to the subproblem corresponding to property j.

No matter which of the four different property scores is used, the definition of the knowledge source cost label for versions D through G reads

$$CK_i^H = \sum_{j \in \mathbf{P}_i^{in}} CP_j^H + c_i.$$

Table 1 presents an overview of all seven versions of the BG-heuristic and it gives a detailed description of all score definitions. Depending on the particular score, the total complexity of the BG-heuristic ranges from $O(np_{max}^{in} + np_{max}^{out} + n^3 + nm) + F$ for Version A to $O(n^2 p_{max}^{in} + nmp_{max}^{in} + n^3 mp_{max}^{out} + n^3 m) + F$ for Version G. Here F represents the complexity of the firing heuristic, which has to be added to the complexity of all BG-heuristics because the information generated by the firing heuristic is required as input for all BG-heuristics. The variation in complexity results from the fact that some of the score definitions consider information that varies from one iteration to another. If this is the case, the knowledge source scores need to be updated in each iteration, causing an increase in complexity. The third column in Table 1 with the heading 'Dyn.' indicates whether scores are updated in each iteration or not.

Table 1. SUMMARY OF ALL BG-HEURISTICS.

Version	Score - CK_i^H	Dyn.	Complexity	$p_{max}^{out} = p_{max}^{in} = O(m)$	$N = max\{n,m\}$
A	c_i	no	$O(np_{max}^{in} + np_{max}^{out} + n^3 + nm) + F$	$O(n^3 + n^2 m)$	$O(N^3)$
B	$c_i / \lvert \mathbf{P}_w^{CT} \cap \mathbf{P}_i^{out} \rvert$	yes	$O(np_{max}^{in} + n^3 p_{max}^{out} + nm) + F$	$O(n^3 m)$	$O(N^4)$
C	$\lvert \mathbf{P}_i^{in} \backslash \mathbf{P}_w^{CT} \rvert - \lvert \mathbf{P}_i^{out} \cap \mathbf{P}_w^{CT} \rvert$	yes	$O(n^3 p_{max}^{in} + n^3 p_{max}^{out} + nm) + F$	$O(n^3 m)$	$O(N^4)$
D	$\bar{c} \sum_{j \in \mathbf{P}_i^{in}} l_j^e + c_i$	no	$O(np_{max}^{in} + n^3 p_{max}^{out} + nm) + F$	$O(n^3 m)$	$O(N^4)$
E	$\bar{c} \sum_{j \in \mathbf{P}_i^{in}} (l_j^e / \mathbf{S}_j) + c_i$	no	$O(np_{max}^{in} + n^3 p_{max}^{out} + nm) + F$	$O(n^3 m)$	$O(N^4)$
F	$\bar{c} \sum_{j \in \mathbf{P}_i^{in}} (l_j^e / \bar{\mathbf{S}}_j) + c_i$	yes	$O(n^2 p_{max}^{in} + n^3 p_{max}^{out} + nm) + F$	$O(n^3 m)$	$O(N^4)$
G	$\sum_{j \in \mathbf{P}_i^{in}} E_j^H + c_i$	no	$O(n^2 p_{max}^{in} + nmp_{max}^{in} + n^3 mp_{max}^{out} + n^3 m) + F$	$O(n^3 m^2)$	$O(N^5)$

2.4 Forward Greedy Heuristic

The third type of a construction heuristic is a solution procedure with a forward search, meaning that the heuristic starts constructing the solution at the first instead of the last layer. The selection criterion is determined in two consecutive steps. First, the heuristic identifies the set $\mathbf{K}_w^{CA}$ of *currently selectable* knowledge sources. Currently selectable are all knowledge sources for which all required input properties are known at the beginning of iteration w, and that have not already been added to the solution in any of the preceding iterations,

$$\mathbf{K}_w^{CA} = \{k_i | \mathbf{P}_i^{in} \subseteq \mathbf{P}_w^{CA}, k_i \notin \mathbf{K}_w^s\}.$$

If there is a feasible solution, $\mathbf{K}_w^{CA}$ can never be empty. The second step considers two cases.

If there is at least one knowledge source in $\mathbf{K}_w^{CA}$ that provides one or more of the current target properties $\mathbf{P}_w^{CT}$, all knowledge sources that do not provide at least one target property are removed from $\mathbf{K}_w^{CA}$. If we denote the modified set of currently selectable knowledge sources by $\mathbf{K}_w^{CAT}$, the selection criterion becomes

$$\mathbf{K}_w^{SI} = \underset{k_i \in \mathbf{K}_w^{CAT}}{\text{argmin}} \{CK_i^H\}. \quad \text{(Case A selection scenario)}$$

If $\mathbf{K}_w^{CA}$ does not contain any knowledge source that provides at least one of the current target properties, the selection criterion reads

$$\mathbf{K}_w^{SI} = \underset{k_i \in \mathbf{K}_w^{CA}}{\text{argmin}} \{C\mathbf{K}_i^H\}. \quad \text{(Case B selection scenario)}$$

Since knowledge sources can only be selected if all input properties are available, the forward greedy (FG) heuristic can never lead to a set of selected knowledge sources with a deadlock.

Three alternative versions of the FG-heuristic are being considered. The different versions vary only in the definition of the knowledge source scores for the Case B selection criterion. As far as the selection criterion for Case A is concerned, all implementations use $CK_i^H = c_i / |P_i^{out} \cap P_w^{CT}|$. Other choices for the score, such as simply c_i or $c_i / |\mathbf{P}_i^{out}|$ are ignoring readily available information and are not considered. Table 2 provides a summary of the different versions of the FG-heuristic, including the varying definition of scores for the Case B selection criterion. Version A considers the basic knowledge source cost labels, $CK_i^H = c_i$. Version B tries to improve on this, by considering the number of output properties generated by the knowledge source, $CK_i^H = c_i / |\mathbf{P}_i^{out}|$. Finally, Version C divides the cost of knowledge source k_i by the number of its output properties that are at the same time input properties to at least one knowledge source k_t, which returns at least one of the current target properties. In essence, we do a one layer forward look up. In order to state this more formally, let $\mathbf{P}_{CT}^{in}$ be the set of all the input properties of all knowledge sources that provide at least one of the current target properties. The knowledge score definition for Case B selection criterion of Version C then reads $CK_i^H = c_i / |\mathbf{P}_i^{out} \cap \mathbf{P}_{CT}^{in}|$. Depending on the score

definition, the worst case complexity of the FG-heuristic varies between $O(n^2 p^{in}_{max} + np^{out}_{max})$ and $O(n^2 p^{in}_{max} + n^2 p^{out}_{max})$. The summary of all FG-heuristics is given in Table 2.

Table 2. SUMMARY OF ALL FG-HEURISTICS.

Version	Case B Score CK^H_i	Dyn.	Complexity	$p^{out}_{max} = p^{in}_{max} = O(m)$	$N = max\{n,m\}$
A	c_i	no	$O(n^2 p^{in}_{max} + np^{out}_{max})$	$O(n^2 m)$	$O(N^3)$
B	$c_i/\|\mathbf{P}^{out}_i\|$	no	$O(n^2 p^{in}_{max} + np^{out}_{max})$	$O(n^2 m)$	$O(N^3)$
C	$c_i/\|\mathbf{P}^{out}_i \cap \mathbf{P}^{in}_{CT}\|$	yes	$O(n^2 p^{in}_{max} + n^2 p^{out}_{max})$	$O(n^2 m)$	$O(N^3)$

2.5 Shortest Path Heuristic

Heuristics based on a repeated application of shortest path algorithms have successfully been applied to many *NP*-complete network problems including the Steiner network problem, see e.g. Hwang *et al.* (1992). Here we present two families of the shortest path heuristic and within each family several variants based on different knowledge source scores.

We first introduce an auxiliary network, called the *ks-ks network*, which serves as the underlying network for shortest path. The nodes correspond to knowledge sources. There is an arc from $k_i \in \mathbf{K}$ to $k_{\bar{i}} \in \mathbf{K}$ if and only if $P^{out}_{k_i} \cap P^{in}_{k_{\bar{i}}} \neq \varnothing$. The cost of this arc corresponds to the score $CK^H_{\bar{i}}$ of $k_{\bar{i}}$. Although the integrity of the CSI-specific precedence constraint information is lost in the ks-ks network, this network does provide the information required for the design of shortest path heuristics. It is important to note that a valid alternative to the presented ks-ks network would be a similar property to property network, which could be obtained by eliminating the knowledge source nodes instead of the property nodes.

The presented shortest path (SP) heuristic adds exactly one new knowledge source to the solution in each iteration of the general framework, i.e. $|K^{SI}_w| = 1$ for every w. In order to find the most suitable knowledge source to be added in a given iteration, the heuristic determines a path Q in the ks-ks network with the smallest total score among all paths from a knowledge source $k_i \in \mathbf{K}^{CA}_w$ to a knowledge source $k_j \in \mathbf{K}^{CT}_w$, where $\mathbf{K}^{CT}_w = \{k_j | \mathbf{P}^{out}_j \cap \mathbf{P}^{CT}_w \neq \varnothing\}$. Two different definitions of the total score of a path are considered: (1) the sum of all arc scores on the path, or (2) the sum of all arc scores on the path divided by the number of target properties provided by the last knowledge source on the path. The second definition of the total score of a path seems more intuitive because it prices the paths on a *per target property* bases. The second pricing strategy, however, comes at an increased complexity. When the first of the two strategies is used, two dummy nodes are added to the ks-ks network and the problem of identifying the knowledge source that should be selected in the current iteration reduces to solving a single s-t shortest path problem as shown in Figure 7. Source s is connected to each knowledge source $k_i \in \mathbf{K}^{CA}_w$ and each knowledge source $k_j \in \mathbf{K}^{CT}_w$ is connected to the sink t. Once the shortest s-t path has been determined, the knowledge source that is added to the solution in the current iteration is simply the second knowledge source on the path with the smallest total path score, i.e., the first knowledge source on the shortest path after the source.

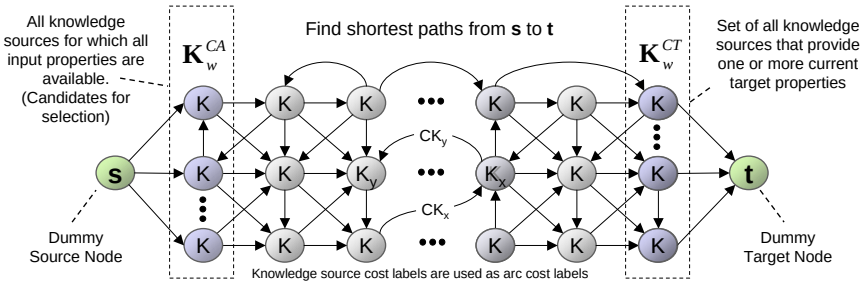

Figure 7. TOTAL PATH SCORE DEFINITION RESULTING IN THE SINGLE S-T SHORTEST PATH PROBLEM.

If the second definition for total score is used, only dummy source s is introduced and a total number of $|\mathbf{K}^{CT}_w|$ s-t shortest path problems have to be solved, see Figure 8. Source s is added as above. Once each of the $|\mathbf{K}^{CT}_w|$ s-t shortest path problems from s to each

of the knowledge sources in $\mathbf{K}_w^{CT}$ is solved, the total score of each shortest path is divided by the number of target properties returned by the last knowledge source on the path. This provides the desired per target property score for each path. The knowledge source that is added to the solution in the current iteration is simply the second knowledge source of the path with the smallest per target property cost.

Five suitable knowledge source score definitions are considered for the SP-heuristic. Selected score definitions used previously with the BG- and the FG-heuristics cannot be applied to the SP-heuristic. The score used in Version F of the BG- heuristic, for example, would not work for an SP-heuristic because the definition of the sets $\overline{\mathbf{S}}_j$ only makes sense for heuristics with a backward search. Considering the two alternative definitions of total score of a path, the five score definitions result in ten different versions of the SP-heuristic. Table 3 summarizes the main characteristics of the different versions, including score definitions, number of s-t shortest path problems solved per iteration, and worst case time complexity. The worst case time complexity of the SP-heuristic is a function of the running time of the shortest path algorithm used as a subroutine within the SP-heuristic. The complexity of this subroutine is denoted by $O(g(k))$, if k is the number of nodes in the network (see e.g. Ahuja *et al.* (1993) for possible shortest path variants).

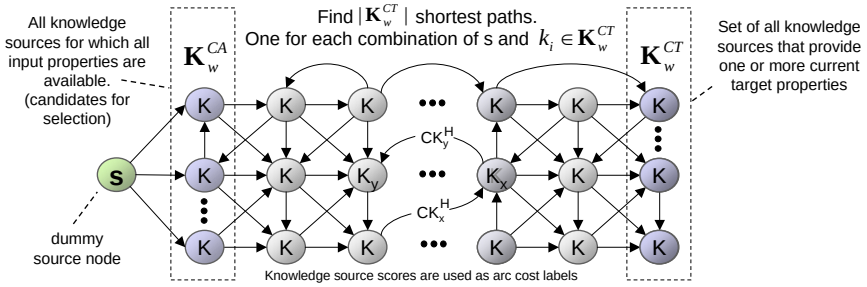

Figure 8. TOTAL PATH SCORE DEFINITION RESULTING IN THE MULTIPLE S-T SHORTEST PATH PROBLEMS.

Table 3. SUMMARY OF ALL SP-HEURISTICS.

Version	Score CK_i^H	Dyn.	No. of s-t paths	Complexity	$p_{max}^{out} = p_{max}^{in} = O(m)$	$N = max\{n,m\}$
A	c_i	yes	1	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n\,g(n))$	$O(n^2 m + n\,g(n))$	$O(N^3 + N\,g(n))$
B		yes	$\lvert K_w^{CT}\rvert$	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n^2\,g(n))$	$O(n^2 m + n^2\,g(n))$	$O(N^3 + N^2\,g(n))$
C	$\overline{c}\lvert\mathbf{P}_i^{in} \setminus \mathbf{P}_w^{CA}\rvert + c_i$	yes	1	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n\,g(n))$	$O(n^2 m + n\,g(n))$	$O(N^3 + N\,g(n))$
D		yes	$\lvert K_w^{CT}\rvert$	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n^2\,g(n))$	$O(n^2 m + n^2\,g(n))$	$O(N^3 + N^2\,g(n))$
E	$\overline{c}\sum_{j\in\mathbf{P}_i^{in}} l_j^e + c_i$	yes	1	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n\,g(n))$	$O(n^2 m + n\,g(n))$	$O(N^3 + N\,g(n))$
F		yes	$\lvert K_w^{CT}\rvert$	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n^2\,g(n))$	$O(n^2 m + n^2\,g(n))$	$O(N^3 + N^2\,g(n))$
G	$\overline{c}\sum_{j\in\mathbf{P}_i^{in}} l_j^e/\mathbf{S}_j + c_i$	yes	1	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n\,g(n))$	$O(n^2 m + n\,g(n))$	$O(N^3 + N\,g(n))$
H		yes	$\lvert K_w^{CT}\rvert$	$O(n^2 p_{max}^{in} + n^2 p_{max}^{out} + nm + n^2\,g(n))$	$O(n^2 m + n^2\,g(n))$	$O(N^3 + N^2\,g(n))$
I	$\sum_{j\in\mathbf{P}_i^{in}} E_j^H + c_i$	yes	1	$O(n^4 p_{max}^{in} + n^4 p_{max}^{out} + n^2 m + n\,g(n))$	$O(n^4 m + n\,g(n))$	$O(N^5 + N\,g(n))$
J		yes	$\lvert K_w^{CT}\rvert$	$O(n^4 p_{max}^{in} + n^4 p_{max}^{out} + n^2 m + n^2\,g(n))$	$O(n^4 m + n^2\,g(n))$	$O(N^5 + N^2\,g(n))$

3 Local Improvement Heuristics

Local improvement heuristics try to discover improved solutions in the vicinity (neighborhood) of already known solutions. One of the most important aspects of a neighborhood search algorithms is the definition of an appropriate neighborhood function. The larger the

neighborhood, the more likely it is that the heuristic discovers the global optimum or at least succeeds in finding a solution that is closer to the global optimum than the initially known solution. Unfortunately, a larger neighborhood generally also results in an increased time complexity. Hence, enlarging the size of the neighborhood does not necessarily result in a more effective heuristic.

Here we present several local search heuristics. As with construction heuristics, we first propose a general framework for the CSI problem and then we give details about five specific heuristics.

3.1 General Framework

To simplify the discussion of local improvement heuristics for the CSI problem a general framework is developed (see Figure 9). All local improvement heuristics for the CSI problem start with a feasible solution (*original solution*) and try to improve upon this solution in an iterative fashion. Given an original solution $\mathbf{K}^S$ of length L, a neighborhood is fully defined by two layers ℓ_1 and ℓ_2 of $\mathbf{K}^S$, with $1 \leq \ell_1 < \ell_2 \leq L$. The two layers control the size of the neighborhood. We can think of them as imposing boundaries on the neighborhood since we try to improve the solution only in layers ℓ_1 through ℓ_2. Depending on the number of layers between ℓ_1 and ℓ_2 we distinguish between single ($\ell_2 = \ell_1 + 1$) and multi-layer ($\ell_2 > \ell_1 + 1$) neighborhoods. The set of all properties that are known in layer ℓ of the original solution are denoted by $\mathbf{P}_\ell^{CAL}$. Given ℓ_1 and ℓ_2 and the corresponding sets of known properties in the two layers, we define a new CSI subproblem instance $\hat{G}(\mathbf{P}, \mathbf{K}, \mathbf{A}, c)$. The definition of the set of the initially known properties $\hat{\mathbf{P}}^I$ and the definition of the set of target properties $\hat{\mathbf{P}}^T$ in $\hat{G}$ depend on the particular design of the local improvement heuristic and is described on an individual bases in sections that follow. The subproblem can be solved either by a construction heuristic presented in Section 2 or by an exact methodology from Bless *et al.* (2006). Note that by definition $\hat{G}$ is feasible. If the total cost of the obtained feasible set $\hat{\mathbf{K}}^S$ to $\hat{G}$ is smaller than the sum of the costs of all knowledge sources $k_i \in \mathbf{K}^S$ used between layers ℓ_1 and ℓ_2 in the original solution, an improvement has been found. The feasible set of the improved solution is

$$\mathbf{K}^S = \hat{\mathbf{K}}^S \cup \left(\mathbf{K}^S \setminus \{\mathbf{K}_{\ell_1}^{SIL}, \mathbf{K}_{\ell_1+1}^{SIL}, ..., \mathbf{K}_{\ell_2}^{SIL}\} \right),$$

where $\mathbf{K}_\ell^{SIL}$ is the set of all knowledge sources that are active in layer ℓ of the original solution $\mathbf{K}^S$. Next we elaborate on several resulting heuristics.

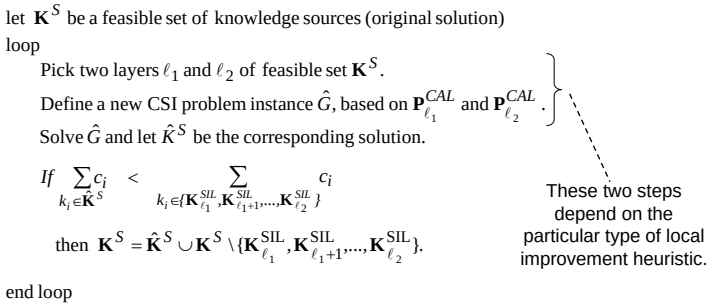

Figure 9. GENERAL FRAMEWORK FOR LOCAL IMPROVEMENT HEURISTICS.

3.2 One-Layer-Check Heuristic

The first local improvement heuristic for the CSI problem, referred to as the one-layer-check (OLC) heuristic, considers a relatively small neighborhood that only allows changing a single layer of the current solution at a time. In OLC, we impose $\ell_2 = \ell_1 + 1$ (see Figure 10). Starting at the first layer ($\ell_1 = 1$) the heuristic attempts to modify each layer of the existing solution. The set of initially known properties and the set of target properties in $\hat{G}$ for each subproblem instance are

$$\begin{aligned} \hat{\mathbf{P}}^I &= \mathbf{P}_{\ell_1}^{CAL} \\ \hat{\mathbf{P}}^T &= \left(\mathbf{P}_{\ell_2}^{R+} \cup \mathbf{P}_N^T \right) \setminus \mathbf{P}_{\ell_1}^{CAL}, \end{aligned} \tag{2}$$

where $\mathbf{P}_N^T$ is the subset of the original target properties $\mathbf{P}^T$ including all target properties that are generated between ℓ_1 and ℓ_2 in the original solution, $\mathbf{P}_{\ell_1}^{CAL}$ represents the set of properties known in layer ℓ_1 of the original solution, and $\mathbf{P}_{\ell_2}^{R+}$ is defined as the set of all input

properties required by the knowledge sources of the remainder of the original solution beyond ℓ_2. The remainder includes all knowledge sources that are used past layer ℓ_2 in the original solution. However, it is important to emphasize that $\mathbf{P}^{R+}_{\ell_2}$ does not include properties that are first generated and then used in the remainder of the solution in a sequential way. For example, a property that is returned as output by a knowledge source in layer $\ell_2+\alpha$, $\alpha \geq 1$ and used as input for a knowledge source in layer $\ell_2+\alpha+1$ is not part of $\mathbf{P}^{R+}_{\ell_2}$.

In case the heuristic finds an improvement, the solution is updated according to the general framework, ℓ_1 remains at its current value, the sets $\mathbf{P}^{CAL}_{\ell}$ and and $\mathbf{K}^{SIL}_{\ell}$ are updated to reflect the changes made to the original solution, and the heuristic is repeated at the same layer. Once no further improvement can be made at the given layer, ℓ_1 is incremented by one. The heuristic terminates when ℓ_1 is equal to L.

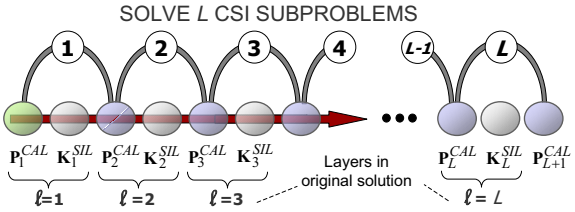

Figure 10. THE ONE-LAYER-CHECK HEURISTIC.

3.3 Forward Wave Heuristic

The second local improvement heuristic considers variable size local search neighborhoods and is referred to as the forward wave (FW) heuristic. While ℓ_2 is equal to L for all subproblems solved over the course of the heuristic, ℓ_1 is incremented gradually from 1 to L. As illustrated in Figure 11, this results in a series of subproblem instances of decreasing neighborhood size. The definitions for the set of initially known properties and the set of target properties for each subproblem instance are

$$\hat{\mathbf{P}}^I = \mathbf{P}^{CAL}_{\ell_1}$$
$$\hat{\mathbf{P}}^T = \mathbf{P}^T \backslash \mathbf{P}^{CAL}_{\ell_1}.$$

3.4 Backward Wave Heuristic

The backward wave (BW) heuristic can be described as a mirror design of the FW-heuristic. Neighborhood boundary ℓ_1 is held constant and the value of ℓ_2 changes from one iteration to the next. More specifically, $\ell_1 = 1$ for all subproblems and ℓ_2 decreases from L to 1. For each subproblem the set of initially known properties and the set of target properties are defined as

$$\hat{\mathbf{P}}^I = \mathbf{P}^{CAL}_{\ell_1},$$
$$\hat{\mathbf{P}}^T = \left(\mathbf{P}^{R+}_{\ell_2} \cup \mathbf{P}^T_N\right) \backslash \mathbf{P}^{CAL}_{\ell_1}.$$

Figure 12 illustrates the different subproblems that are solved by the BW-heuristic.

3.5 Split-Layer and Recursive Bisection Heuristics

The basic idea of these local improvement heuristics is to define two CSI subproblems by splitting the original solution in half and by trying to improve each of the two halves separately. Splitting the original solution provides us with two solution subnetworks, one including all knowledge sources used from layer 1 through layer $\lfloor L/2 \rfloor$ and one including all knowledge sources selected in layers $\lfloor L/2 \rfloor + 1$ through L of the original solution. If the splitting operation is performed only once, the heuristic is referred to as the split-layer (SL) heuristic (see Figure 14). If the splitting operation is performed repeatedly, the heuristic is referred to as the recursive bisection (RB) heuristic (see Figure 15). The depth of the recursion is a parameter. If the parameter is equal to one, we obtain the SL-heuristic. Since the heuristic splits the parent problem into two separate subnetworks, two new child problems are defined. The child problem defined on the 'right' subnetwork is always solved first. The rational behind this is the fact that it is easier to maintain feasibility if the existing solution is modified from left to right as opposed to the other way round. Let L^P be the length of the parent solution and let

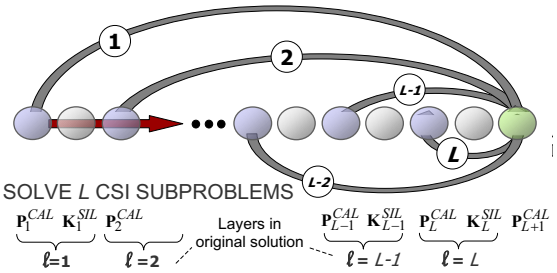

Figure 11. THE FORWARD WAVE HEURISTIC.

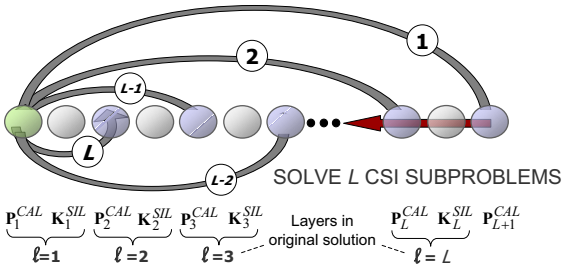

Figure 12. THE BACKWARD WAVE HEURISTIC.

$L^{PM} = \lfloor \frac{L^P}{2} \rfloor$. Given these definitions, the set of initially known properties and the set of target properties for the right child are defined as

$$\hat{\mathbf{P}}^I = \mathbf{P}^{CAL}_{L^{PM}}$$
$$\hat{\mathbf{P}}^T = \left(\mathbf{P}^{R+}_{L^P} \cup \mathbf{P}^{RT}_{L^{PM}}\right) \setminus \left(\mathbf{P}^{RT}_{L^P} \cup \mathbf{P}^{CAL}_{L^{PM}}\right),$$

where $\mathbf{P}^{R+}_{L^P}$ is defined in the same way as $\mathbf{P}^{R+}_{\ell_2}$ in the OLC and the BW-heuristic, and $\mathbf{P}^{RT}_{L^{PM}}$ and $\mathbf{P}^{RT}_{L^P}$ are the sets of all target properties of the original problem that are still unknown in layer L^P and L^{PM}, respectively. Note that $\lfloor L^{PM} \rfloor$ represents the left boundary (ℓ_1) and L^P the right boundary (ℓ_2) of the neighborhood of the right child problem.

After solving the subproblem based on the right portion of the parent problem, the problem defined on the left half of the parent problem is solved. The set of initially known properties and the set of target properties for the left child problem are defined as

$$\hat{\mathbf{P}}^I = \mathbf{P}^{CAL}_1$$
$$\hat{\mathbf{P}}^T = \left(\mathbf{P}^{R+}_{L^{PM}} \cup \mathbf{P}^{RT}_1\right) \setminus \left(\mathbf{P}^{RT}_{L^{PM}} \cup \mathbf{P}^{CAL}_1\right).$$

Here the left boundary is 1 (ℓ_1) and the right boundary is L^{PM} (ℓ_2).

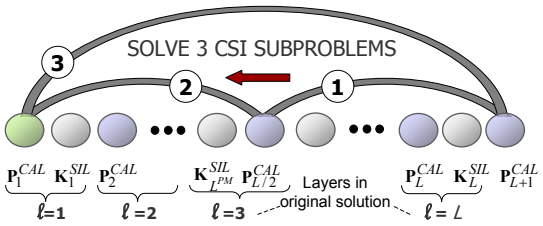

Figure 13.

4 Computational Experiments

The discussion of computational experiments for the developed heuristics is divided into three parts. The first part provides a comparison of the individual performance of construction heuristics. In the second section we evaluate the local improvement heuristics. In the third part the performance of the heuristics is compared to the optimal value. All experiments were performed on an INTEL® Pentium® 4 (3.00GHz) personal computers with 512KB integrated level 2 cache, a bus speed of 533MHz, and 512MB DDR SDRAM-333 MHz system memory. The operating system was Windows XP® and the implementations were done in Matlab® version 6.5. CPLEX 8.0® was used as the integer programming solver for comparison purpose.

A more comprehensive computational study is presented in Bless (2004). Here we present the most significant findings. Bless *et al.* (2006) have shown that property saturation, which is defined as the ratio between the number of knowledge sources and properties ($r^s = \frac{m}{n}$), has a significant effect on the complexity of CSI instances. Considering this observation, property saturation was expected to

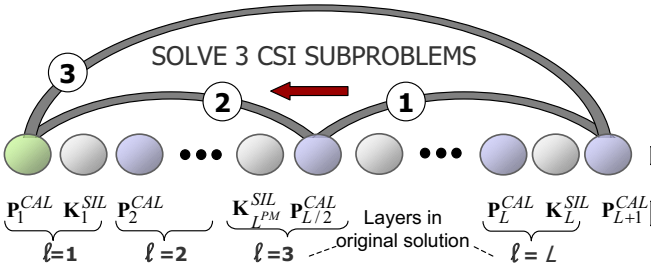

Figure 14. THE SPLIT-LAYER HEURISTIC.

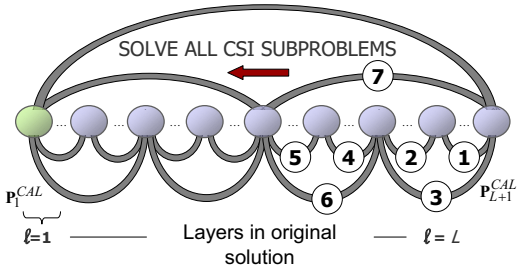

Figure 15. THE RECURSIVE BISECTION HEURISTIC.

have an effect on the performance of heuristic solution procedures as well. Therefore, all developed heuristics were tested at various levels of property saturation, ranging from 10% to 100%. Only in few experiments we keep the saturation fixed and vary the number of knowledge sources.

4.1 Performance Comparison of Construction Heuristics

For the experiments in this subsection the instances were randomly generated and all other network parameters were held constant at $n = 100$, $\tilde{p}^I = 3$, $\tilde{p}^T = 3$, $\tilde{D}^{in} = U(1,3)$, $\tilde{D}^{out} = U(1,3)$, and $c_i = U(1,500)$. Here $U(a,b)$ represents the uniform distribution on integers between a and b. Quantities $\tilde{D}^{in}$, $\tilde{D}^{out}$ represent the knowledge source in- and out-degree distribution, respectively. For example, $\tilde{D}^{in} = U(1,3)$ means that $|\mathbf{P}_i^{in}|$ is distributed based on $U(1,3)$. The knowledge source cost is distributed based on $U(1,500)$. Infeasible instances are not considered. For each data point in the charts and tables that follow the values were obtained as the average over a 1,000 randomly generated instances, unless stated otherwise.

The two firing heuristics do not perform well and therefore they are not considered as stand alone heuristics. The most important performance measure is the gap, which is computed as $100 \cdot (hv - bv)/bv$, where hv is the solution value obtained by the heuristic in question and bv is the value of the best known solution. The latter is obtained by first running all possible heuristics and recording the value of the best solution. Clearly, the lower the gap, the better the heuristic.

4.1.1 Performance of Different Versions of the Backward Greedy Heuristic Table 4 provides a comparison with respect to the average gap of the performance of all seven versions of the BG-heuristic. The results suggest that Version G clearly outperforms all other BG-heuristics at all levels of property saturation. Versions B, D, E, and F all show very similar performance, while Version A and especially Version C are clearly worse compared to all other versions. It is also clear that the BG-heuristic performs better in higher saturation levels.

Table 4. AVERAGE GAP OF THE BACKWARD GREEDY HEURISTIC.

	Saturation										
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
A	54.20	40.58	29.82	22.22	15.87	11.83	8.14	5.57	3.75	3.12	19.51
B	53.10	38.44	28.02	21.00	14.84	10.98	7.36	4.88	3.46	2.80	18.49
C	68.48	45.42	32.62	22.58	16.82	11.30	8.72	5.53	4.00	3.14	21.86
D	53.10	37.22	24.81	17.84	12.84	8.77	6.45	4.17	3.01	2.27	17.05
E	53.10	37.93	25.95	19.04	13.33	9.56	6.95	4.58	3.63	2.69	17.68
F	54.03	39.12	26.59	18.86	14.13	9.58	7.40	4.61	3.77	2.78	18.09
G	50.92	32.61	22.86	15.62	11.68	7.62	5.94	3.72	2.72	2.01	15.57

4.1.2 Performance of Different Versions of the Forward Greedy Heuristic Table 5 presents the same type of analysis for the different versions of the FG-heuristic. The results indicate that Version C outperforms the other versions very decisively at all, but

the smallest levels of property saturation. To the contrary to the BG-heuristic, the gaps for the forward greedy heuristic are significant also for large saturations. By comparing Table 5 and Table 4 it is clear that the BG-heuristic is much more efficient for saturations exceeding 30%. On the other hand, for low saturation instances the FG-heuristic outperforms the BG-heuristic.

Table 5. AVERAGE GAP OF THE FORWARD GREEDY HEURISTIC.

	Saturation										
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
A	18.52	27.06	23.11	27.37	32.61	37.37	38.34	37.77	36.67	34.04	31.29
B	18.52	26.85	21.38	25.36	31.03	35.21	37.21	35.60	34.60	31.44	29.72
C	18.52	26.75	19.48	19.61	21.40	26.24	28.90	29.04	28.51	26.44	24.49

4.1.3 Performance of Different Versions of the Shortest Path Heuristic

Table 6 shows a comparison of the performance of the ten different implementations of the SP-heuristic. Recall that there are only five different score definitions. Hence, there are five pairings (A&B, C&D, E&F, G&H, I&J) sharing the same knowledge source score definition. The difference within each pairing is the definition of total path score as discussed in Section 2.5. Versions G and H are the most robust heuristics. The two of them yield the smallest gap for all saturations except at 40%, where Versions I and J prevail. The conclusion is that Versions G-I are the most effective.

An important observation is the fact that multiple s-t path variants do not consistently outperform their single s-t path counterpart. Due to their higher computational times and this observation, they are no longer considered.

By comparing all three tables, we conclude that on average the SP-heuristic is the most effective heuristic among the three for saturation levels smaller than 50%. Once this saturation level is exceeded, the BG-heuristic becomes more efficient. The good performance of the SP-heuristic for low saturation levels is expected since in these cases the underlying network is sparse and therefore "unconstrained" paths are a good approximation to the paths from the problem definition. We note that this does not imply that the FG-heuristic is outperformed on all of the instances. It turns out that on approximately 4% of the instances, the FG-heuristic found the best solution among the three heuristics.

Table 6. AVERAGE GAP OF SHORTEST PATH HEURISTICS.

	Saturation										
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
A	6.58	13.07	19.72	24.25	27.03	29.15	28.64	28.03	26.67	23.28	22.64
B	7.36	13.01	17.41	20.41	22.12	23.87	23.19	23.30	21.53	19.14	19.13
C	19.01	21.60	17.38	18.73	21.78	24.35	25.25	24.35	21.91	18.21	21.26
D	18.98	20.54	18.44	19.70	21.86	24.99	25.39	24.32	22.23	18.14	21.46
E	6.63	14.77	16.74	18.20	20.57	23.39	23.95	22.36	19.68	15.35	18.16
F	7.85	14.70	17.63	18.75	20.28	23.63	24.33	22.63	19.62	15.20	18.26
G	6.50	10.62	14.05	16.80	19.39	22.98	23.67	23.05	19.53	15.27	17.19
H	7.58	11.82	15.37	18.03	19.16	23.27	23.80	23.04	19.56	15.17	17.68
I	12.26	15.70	14.80	15.53	19.67	23.33	25.32	24.09	21.69	18.18	19.06
J	12.35	15.36	14.63	15.57	20.00	23.89	25.32	24.15	21.82	17.99	19.11

4.2 Comparison of Local Improvement Heuristics

Experiments with varying levels of property saturation were also conducted for the five local improvement heuristics. All considered problem instances presented in this section have the following network characteristics: $n = 50$, $r^s = 1\%$ to 100%, $\tilde{p}^I = 3$, $\tilde{p}^T = 3$, $\tilde{D}^{in} = U(1,3)$, $\tilde{D}^{out} = U(1,3)$, $c_i = U(1,500)$. In order to be able to establish the quality of the produced solutions, we use only 50 knowledge sources as opposed to a 100 in the study of construction heuristics. For 100 knowledge sources it is very difficult to find

optimal solutions. Since local search heuristics perform better for large instances, such a choice only underestimates the quality of our heuristics.

From the previous section it is clear that Version G of the BG-heuristic, Version C of the FG-heuristic, and Version G of the shortest path heuristic are the most efficient. We point out that the running times of any of these heuristics are less than one second per instance. For these reason from now on we consider only these three construction heuristics and we simply denote them as the *BG-heuristic*, the *FG-heuristic*, and the *SP-heuristic*. To test the local improvement heuristics, every one of them was run on each of the three solutions obtained by the three construction heuristics. Furthermore, each of the subproblems that have to be solved within each of the local improvement heuristic was also solved with the three available construction heuristics. This is tractable due to the low execution times of the construction heuristics. The parameter that determines the depth of the recursion in the RB-heuristic is set to 10.

The main findings with respect to the gap are reported in Table 7. The first important observation is that these gaps are much lower than the gaps obtained by the construction heuristics. This implies that local improvement heuristics are efficient. Among the five different local improvement strategies, the recursive bisection RB dominates regardless of the saturation. Most of the heuristics perform much better at the two extreme saturation levels, i.e., very low or high saturation. BW and SL also perform well and the forward wave FW heuristic is inferior to any other heuristic.

Table 7. AVERAGE GAP OF LOCAL IMPROVEMENT HEURISTICS.

	Saturation									
	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
OLC	0.77	1.30	1.24	1.80	2.39	1.94	1.30	1.22	0.82	1.42
SL	0.92	1.37	1.21	1.62	1.98	1.47	1.04	1.14	0.79	1.28
RB	0.63	0.76	0.61	0.57	0.66	0.57	0.38	0.33	0.28	0.53
BW	0.78	1.40	1.20	1.19	1.53	0.85	0.67	0.57	0.36	0.95
FW	2.16	3.13	2.81	2.94	3.17	2.57	2.02	1.55	1.23	2.40

The running times of all heuristics are presented in Table 8. These times include the time required by the three construction heuristics. The most efficient RB-heuristic is unfortunately computationally the most intensive. On the positive note, the SL-heuristic produces good solutions and it is relatively inexpensive from the CPU point of view. While these times range only in seconds, it is demonstrated in Section 4.3.2 that they grow substantially with the increased number of knowledge sources.

Table 8. AVERAGE CPU TIME (in seconds) OF HEURISTICS.

Saturation	OLC	SL	RB	BW	FW
20%	1.32	0.69	1.06	0.59	1.48
30%	1.56	0.81	1.56	0.89	1.79
40%	1.78	0.86	1.97	1.18	2.14
50%	2.17	0.95	2.71	1.71	2.88
60%	2.44	1.02	3.20	2.11	3.49
70%	2.65	1.06	3.59	2.45	3.91
80%	2.78	1.07	3.91	2.79	4.28
90%	2.90	1.10	4.15	3.02	4.63
100%	2.79	1.05	4.08	2.98	4.45
Average	2.27	0.96	2.91	1.97	3.23

From this point on we consider only the three construction heuristics, which are then followed by the backward wave, split layer, and recursive bisection local improvements.

4.3 Heuristics vs. Optimal Solutions

In order to assess the performance and solution quality of the heuristics, additional computational experiments were conducted. We first compare the solution quality with respect to an integer programming formulation (see Bless *et al.* (2006)), and next we discuss significantly larger instances.

4.3.1 Comparison with a Mixed Integer Programming Formulation In these experiments, while property saturation was held constant at 70%, the number of knowledge sources was gradually increased. All other network properties are the same as in the previous section. A sample of 100 randomly generated problem instances was solved at each level of n. The integer programming formulation with a time limit of one hour was used to obtain optimal or near optimal solutions. The considered heuristics are as discussed above. Experiments with varying property saturation were not performed because problem instances for which the linear programming root node relaxations can be obtained are too small to provide meaningful results, i.e., the local improvement heuristics rarely find improvements over the construction heuristics.

The results are demonstrated in Table 9. Whenever CPLEX, which was used to solve the integer programming formulation, was not able to find an optimal solution within an hour, the best lower bound is considered in the gap calculation. The performance of heuristics is excellent. For $n \leq 26$ the obtained solution is always within 2% from the optimal solution (note that CPLEX was able to find an optimal solution in these cases). For the remaining cases we point out that the gaps are with respect to the (linear programming) lower bound, which seems to be very weak. The gaps reported by the heuristics are not substantially larger than CPLEX gaps.

Table 9. GAP OF HEURISTICS VERSUS CPLEX (1 hour limit).

	Number of knowledge sources n						
	10	18	26	34	42	50	Average
BW	1.74%	1.00%	1.45%	2.86%	6.61%	14.01%	4.61%
SL	1.74%	0.94%	0.75%	1.90%	6.28%	12.86%	4.08%
RB	1.74%	0.90%	1.36%	2.62%	6.30%	13.25%	4.36%
CPLEX	0.00%	0.00%	0.00%	0.35%	4.33%	11.62%	2.72%

Another very important aspect is that it took one hour for CPLEX to obtained solutions for $n \geq 34$, while the worst running time of our heuristics is a mere 7 seconds. Next we address much larger instance sizes.

4.3.2 Large-scale Instances Two types of experiments were performed to assess the performance of the developed heuristics on large-scale problem instances ($n > 100$). Note that for these problems we cannot use CPLEX due to excessive computational times. While the first set of experiments is designed to determine the computational limits of the heuristics in terms of problem size, the second set of experiments strives to assess the quality of the heuristic solutions for large-scale instances.

Figure 16 shows the average CPU time required by each construction heuristic for an increasing number of knowledge sources. To show scalability beyond 15,000 knowledge sources, we include Version D of the backward greedy heuristic. Property saturation was held constant at 70% for all instances since it turns out that these are the most challenging instances. Networks with less than 15,000 knowledge sources can be solved in less than an hour with any construction heuristic. It takes the least amount of time for the FG-heuristic, followed by the BG-heuristic, and the SP-heuristic is the slowest one due to the need of computing several shortest paths. If we want to solve even larger instances, than Version D of the backward greedy heuristic need to be employed. It follows from Table 4 that this heuristic is efficient as well. Further computational experiments show that each version of the shortest path heuristic requires approximately the same amount of time as the SP-heuristic, and likewise, all versions of the forward greedy heuristic require the same order of time as the FG-heuristic. We stress at this point that the heuristics were implemented within an interpreter. We believe that these times can be further reduced if the heuristics are implemented within a compiled language.

Due to the fact that all local improvement heuristics presented in this study use the developed construction heuristics as subroutines, the CPU times of the local improvement heuristics are functions of the CPU times of the construction heuristics. Hence, using the information presented in Figure 16 and the particular structure of local improvement heuristics, reliable CPU time estimates can be obtained for these solution procedures. We do not show these results.

Assessing the quality of the solutions obtained from the developed heuristics is difficult because it is generally impossible to solve these instances to optimality in a reasonable amount of time with exact solution procedures. Even obtaining acceptable lower bounds

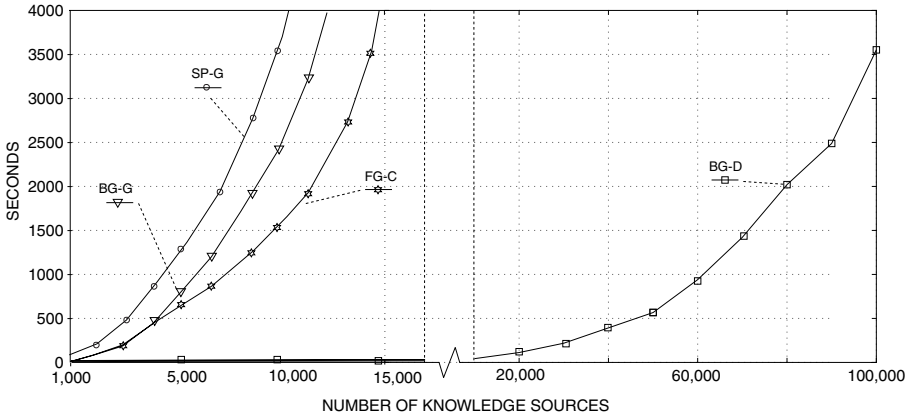

Figure 16. CPU TIMES OF CONSTRUCTION HEURISTICS FOR LARGE-SCALE PROBLEM INSTANCES.

is impractical. To overcome this difficulty, a procedure to generate large problem instances with known optimal solutions has been developed. The idea is to generate random instances with the desired network dimensions and then implant additional feasible solutions into the original network that are guaranteed to be better than all other solutions in the network. The implanted solutions are guaranteed to have lower cost than all other solutions in the network. Using this procedure, problem instances with known optimal solutions and arbitrary size can be generated.

To obtain realistic problem instances, various combinations of three different types of solutions were implanted into the problem instances. Figure 17 shows these three types of solutions. The first is a simple '*p-k-p...-k-p*'–chain that starts at a known property and generates all target properties. The second type of an implanted solution has a more complex topology with knowledge sources featuring multiple in and output properties in the center portion of the solution. More specifically, the first five layers of the solution always look exactly like the solution depicted in Figure 17. The remainder of the solution is a simple '*p-k-p...-k-p*'–chain. The length of the single chain is determined by the desired total length of the solution and the target properties are positioned as shown. Finally, the third type of solutions is composed of three disjoint '*p-k-p...-k-p*'–chains that eventually merge to a single chain that then provides all target properties. All three types of solutions are parameterized and have a variable length portion in the middle of the solution. This allows for the generation of variable length implanted solutions. The cost for the implanted knowledge sources are generated randomly within ranges that guarantee the implanted solutions to be optimal.

A wide range of experiments were performed using different combinations and varying numbers of implanted solutions. Figure 18 shows the average optimality gap of the construction heuristics for a subset of these experiments. Value F is the minimum length of any feasible solution, which is obtained by the firing heuristic. The number of knowledge sources is varied between 1000 and 2000, while the property saturation is held constant at 70%. All other network characteristics are the same as in the experiments described above. The results suggest that the construction heuristics are able to find very good solutions even for large problem instances. For all experiments, the average optimality gap, considering always the best solution found among all three construction heuristics, never exceeded 30% and usually lies between 0 and 20%. Since this is about the same gap that was observed for small problem instances, this could suggest that the performance of the heuristics is insensitive to the instance size. The results for the combination of implanted solutions that was found to lead to the largest optimality gaps are depicted in Figure 18-(e). These problem instances include one implanted solution of Type 1 with a length of $F+2$ and one implanted solution of Type 2 with $F+5$ layers. In an attempt to create hard problem instances, the topologically simpler and also shorter solution of Type 1 was intentionally made more expensive than the Type 2 solution. However, even for these supposedly difficult instances, the performance of the construction heuristics lies within acceptable limits. Moreover, if the same problem instances are solved with both construction and the split-layer local improvement heuristic, the average gap can be reduced to less than 10% as shown on Figure 18-(f). This confirms, the good performance of the SL-heuristic observed for small problem instances. The RB-heuristic performs slightly better than the SL-heuristic but requires much longer CPU times for problem instances beyond 1000 knowledge sources and a property saturation of 70%. Hence, only the SL-heuristic was used to obtain the data shown in Figure 18-(f). Considering time complexity and average improvement, the SL-heuristic can be considered to be the most efficient local improvement heuristic.

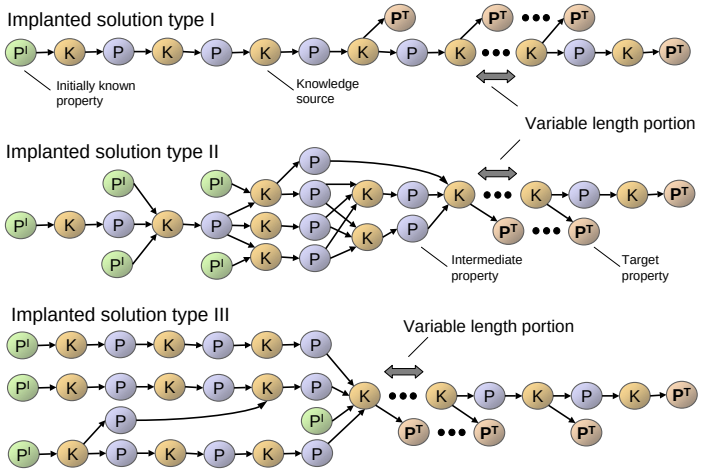

Figure 17. DIFFERENT TYPES OF IMPLANTED SOLUTIONS .

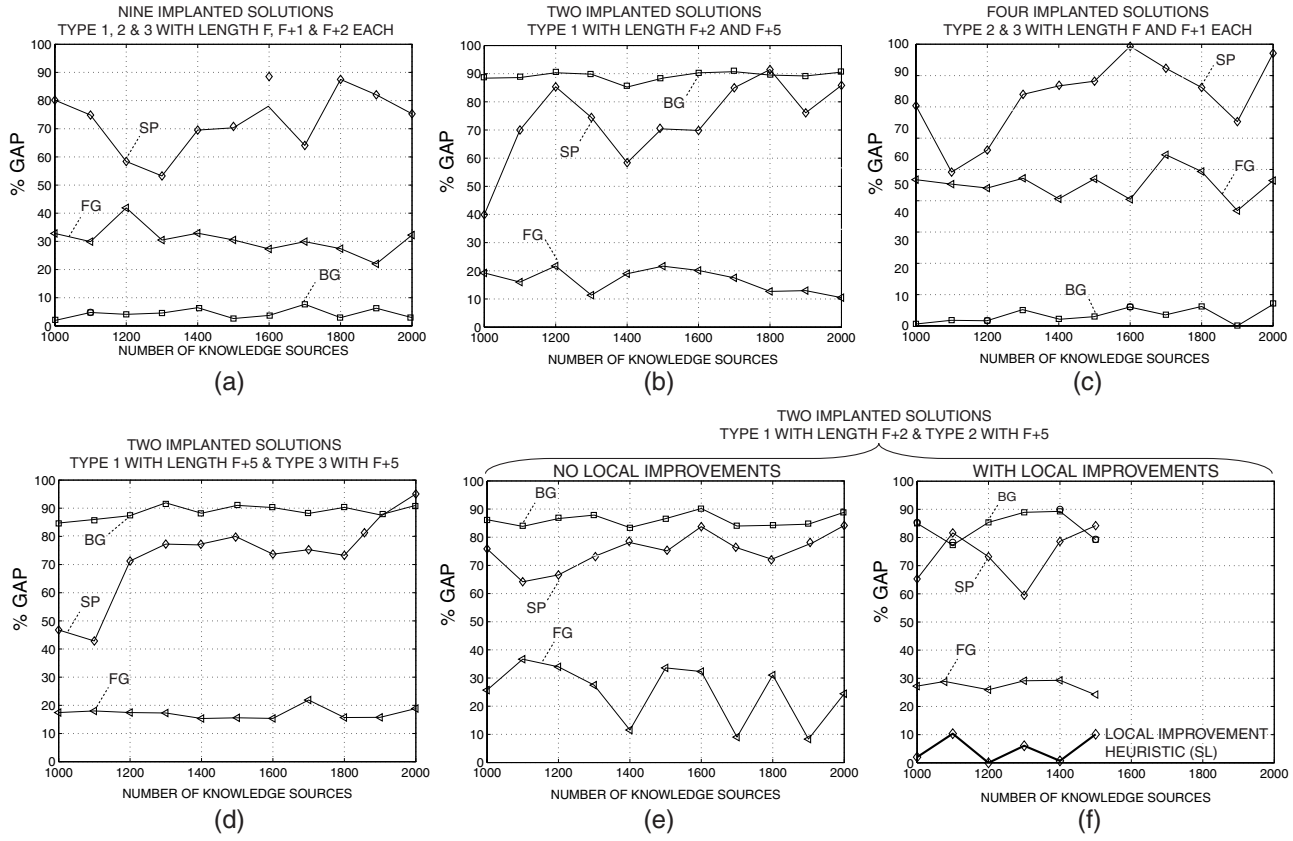

Figure 18. AVERAGE OPTIMALITY GAPS FOR LARGE-SCALE PROBLEM INSTANCES WITH VARIOUS COMBINATIONS OF IMPLANTED SOLUTIONS.

5 Conclusions and Future Work

A general framework for the development of construction heuristics for the CSI problem has been presented. Using this framework four different types of heuristics have been developed, including a basic search algorithm (the firing heuristics), two greedy heuristics (backward greedy and forward greedy heuristic), and a shortest-path heuristic. Each of the four types of construction heuristics has been implemented using different score definitions.

A general framework for the development of local improvement heuristics for the CSI problem has been presented and five different local improvement heuristics have been discussed. Each heuristic has a varying size search neighborhood. Based on computational complexity and average relative improvement, the split-layer heuristic was found to be the most efficient type of a local improvement heuristic.

Computational experiments have been conducted in order to evaluate and compare the performance of the various heuristic solution procedures. Based on the execution times and the optimality gap, the presented heuristics provide a very competitive solution strategy even for relatively small problem instances, where exact solution procedures are generally expected to dominate. Using specially designed problem instances we were able to assess the solution quality of the heuristics for large-scale instances. Even on very large networks, construction and local improvement heuristics combined, were able to find near optimal solutions for most problem instances.

Based on all computational experiments the following recommendations for solving CSI problem instances are provided.

1. Based on property saturation:
   - Problem instances with a property saturation of 40% or less should always be solved with the SP-heuristic first. If additional CPU time is available, the remaining two heuristics can be run next.
   - Problem instances with a property saturation of 50% or more should be solved with the BG-heuristic.
2. Based on problem size:
   - For problem instances with less than 50 knowledge sources it is worthwhile to make a solution attempt using the exact solution procedures presented by Bless *et al.* (2006).
   - A problem instance with 50-15,000 knowledge sources can be solved within reasonable time limits with all developed construction heuristics. Hence, the selection of a suitable array of construction heuristics should be based on saturation. The best solutions found by the selected array of construction heuristics should then be improved using either the SL or the RB local improvement heuristic.
   - Problem instances with more than 20,000 knowledge sources should be solved with Version D BG-heuristic. All other construction heuristics result in excessive execution times. The SL local improvement heuristic can be used to improve upon the best solution found by this construction heuristic.

Regarding future work, we propose two enhancements. One shortcoming of all heuristic approaches presented in this study, is their inability to continue the search upon becoming trapped in a local optimum. The local improvement heuristics circumvent this problem to a certain degree but eventually these techniques can get trapped in a local optimum as well. This suggests other heuristic techniques such as tabu-search, simulated annealing, genetic algorithms, and meta heuristics. Randomization of the developed heuristics is another possible approach (GRASP).

If in a given knowledge domain several CSI problems have to be solved repeatedly, the development of suitable learning concepts including the design of intelligent feedback channels can be extremely valuable. Given the characteristics of the CSI problem, including its unique network structure and the usually very heterogeneous set of knowledge sources, both inductive and analytical learning approaches can be pursuit. The development of such methods will eventually lead to domain-specific *CSI search engines* that are able to autonomously evolve as more knowledge sources and properties are added to the domain.

Acknowledgements The funding for this project is provided by the National Science Foundation under Grant NSF DMI-00-04226.

We are also extremely thankful to an anonymous referee for providing constructive comments, which lead to a substantially improved manuscript.

REFERENCES

Aarts, E. and Lenstra, J. (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, New York, NY.

Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows*. Prentice Hall, Upper Saddle River.

Ahuja, R., Ergun, O., Orlin, J., and Punnen, A. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, **123**, 75–102.

Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Strublen, S., Takacsi-Nagy, P., Trickovic, I., and Zimek, S. (2002). Web service choreography interface (WSCI) 1.0. Technical report, W3C. http://www.w3.org/TR/WSCI.

Bakken, D. (2001). *Middleware*. Kluwer Academic Press, Norwell, MA.

Bar-Yehuda, R. and Even, S. (1981). A linear-time approximation algorithm for the weighted vertex cover problems. *Journal of Algorithms*, **2**, 198–203.

Bless, P. (2004). *Automated knowledge source selection and service composition in manufacturing*. Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL.

Bless, P., Klabjan, D., and Chang, S. (2006). Automated Knowledge Source Integration and Service Composition. Technical report, University of Illinois at Urbana-Champaign. Available at http://netfiles.uiuc.edu/klabjan/www or http://www.knowledgesources.net.

Chandrasekaran, B., Josepson, J., and Benjamins, V. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems*, **14**, 20–26.

Cheng, H. C. and Jeng, J.-J. (1997). Reusing analogous components. *Knowledge and Data Engineering*, **9**, 341–349.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web services description language 1.1 (WSDL). Technical report, W3C. http://www.w3.org/TR/WSDL.

Coppersmith, D. and Vishkin, U. (1985). Solving NP-hard problems in 'almost trees' : vertex cover. *Discrete Applied Mathematics*, **10**, 27–45.

Dror, M. and Haouari, M. (2000). Generalized Steiner problems and other variants. *Journal of Combinatorial Optimization*, **4**, 415–436.

Engevall, S., Goethe-Lundgren, M., and Vaerbrand, S. A. (1998). Strong lower bound for the node weighted Steiner tree problem. *Networks*, **31**, 11–17.

Fernau, H. and Niedermeier, R. (2001). An efficient exact algorithm for constraint bipartite vertex cover. *Journal of Algorithms*, **38**, 374–410.

Fuchs, B. (2003). A note on the terminal Steiner tree problem. *Information Processing Letters*, **87**, 219–220.

Garey, M. and Johnson, D. (1979). *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, NY.

Glover, F. and Kochenberger, G. (2002). *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA.

Groetschel, M., Martin, A., and Weismantel, R. (1997). The Steiner tree packing problem in VLSI design. *Mathematical Programming*, **78**, 265–281.

Halperin, E. and Srinivasan, A. (2002). Improved approximation algorithms for the partial vertex cover problem. *Lecture notes in Computer Science*, **2462**, 161–174.

Hwang, F. K., Richards, D. S., and Winter, P. (1992). *The Steiner Tree Problem*. Annals of Discrete Mathematics. North-Holland, Amsterdam.

Kolen, A. and Pesch, E. (1994). Genetic local search in combinatorial optimization. *Discrete Applied Mathematics*, **48**, 273–284.

Lawrence, J. (1978). Covering the vertex set of a graph with subgraphs of smaller degree. *Discrete Mathematics*, **21**, 61–68.

Lucena, A. and Beasley, J. E. (1998). A branch and cut algorithm for the Steiner problem in graphs. *Networks*, **31**, 39–59.

Maculan, N., Souza, P., and Vejar, A. C. (1991). An approach for the Steiner problem in directed graphs. *Annals of Operations Research*, **33**, 471–480.

McIlraith, S. (2001). Semantic web services. *IEEE Intelligent Systems*, **16**, 46–53.

Meek, D. L. and Parker, R. G. (1994). A graph approximation heuristic for the vertex cover problem on planar graphs. *European Journal of Operational Research*, **72**, 588–597.

Niedermeier, R. and Rossmanith, P. (2003). On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, **47**, 63–77.

Peltz, C. (2003). Web services orchestration - A review of emerging technologies, tools, and standards. Technical report, Hewlett Packard.

Plesnik, J. (2001). Minimum cost edge subset covering exactly k vertices of a graph. *Journal of Combinatorial Optimization*, **5**, 275–286.

Rosenwein, M. B. and Wong, R. T. (1995). A constrained Steiner tree problem. *European Journal of Operational Research*, **81**, 430–439.

Staab, S. (2003). Web services: Been there, done that? *IEEE Intelligent Systems*, **18**, 72–85.

Steinhöfel, K., Albrecht, A., and Wong, C. K. (2003). An experimental analysis of local minima to improve neighbourhood search. *Computers and Operations Research*, **30**, 2157–2173.

Verhoeven, M. G. A. (1999). Parallel local search for Steiner trees in graphs. *Annals of Operations Research*, **90**, 185–202.

Voss, S. (1999). The Steiner tree problem with hop constraints. *Annals of Operations Research*, **86**, 321–346.

Zachariasen, M. (1999). Local search for the Steiner tree problem in the Euclidean plane. *European Journal of Operational Research*, **119**, 282–300.