H-0212 November 24, 2003
Computer Science

IBM Research Report

Compact Representations of Search in Complex Domains

Yosi Ben-Asher
CS Department
Haifa University
Haifa Israel

Eitan Farchi
IBM Research Division
Haifa Research Laboratory
Haifa 31905, Israel

Compact Representations of Search in Complex Domains

Yosi Ben-Asher
CS Department Haifa University
Haifa, Israel.
yosi@mathcs2.haifa.ac.il

Eitan Farchi
I.B.M Research Center
Haifa, Israel.
efarchi@vnet.ibm.com

November 17, 1998

Abstract

We consider the problem of searching in complex domains for a buggy element. The structure of the domain determines the set of possible queries at every stage of the search. An important type of search domain is a Directed Acyclic Graph (DAG), where one can query every sub-graph.

In this work we focus on reducing the number of sets required to represent a search domain. Our results include:

- An exact definition of general search domains as a set of sets.
- Modeling searching as a two player matrix game, in which a recursive representation allows us to apply backward induction to tackle the size representation problem.
- The set representation is powerful enough to represent almost any type of search; however, this generality usually requires exponential sizes. As it turns out, some specific types of search domains (like searching in DAGs) can be represented more compactly by using the original graph instead of sets. It is therefore important to determine which search domains (represented as sets) are actually isomorphic to a search in a DAG. We find necessary and sufficient conditions for representing search domains as DAGs, and show explicit constructions for transforming such search domains into DAGs.

Search problems are often encountered in program testing or debugging. The bug should be found by searching the program's control graph using a minimal set of queries. Search problems also appear in the area of searching classified large tree-like data bases (e.g. the Yahoo's data base of the Internet).

1 Introduction

We address the problem of "searching in domains" in order to locate a buggy element. The domain itself is structured such that we can query parts of it, and, if the queried part contains a buggy element, continue searching in that part alone. If the answer to the query is negative, then the part has no buggy elements, and the searching process continues checking the complementary part of the domain. A classical example is the search for a "marked" number in the set $[1, \ldots, n]$. A query $i \in [1, \ldots, n]$ returns 'yes' if the number is smaller than i, and the search continues with the range $[1, \ldots, i-1]$. Otherwise, it continues with the range $[i, \ldots, n]$. In this case the optimal strategy is to use a binary search and query $i = \frac{n}{2}$.

We use set of sets to describe the structure of general search domains. Denote the set of elements of the search domain by D. For any subset of the search domain $\alpha_i \subset D$, such that there is a history of the search process that reaches α_i, we postulate a set $S_{\alpha_i} = \{\beta^i_1, ..., \beta^i_{i_k}\}$, $\beta^i_j \subset \alpha_i$ of subsets of α_i. The β^i_js can be used by the searcher to search for the buggy element in α_i when α_i is reached during the search process.

The expressive power of the above set representation is evidently high. The requirement to continue the search with a subset or its complement, when the answer to the query is no, does not restrict its expressive power. It is simply a way of formalizing that progress has been made in the search. However, the cost of the set representation in terms of space complexity is too high, as the number of sets involved with the representation is usually exponential in the size of the domain D.

We consider two ways of reducing the effect of the size representation problem.

First, we consider the search process as a two player game (a hider and a searcher). We want to compute the Nash equilibrium of the game (in pure strategies). The matrix game derived from the set representation of the search domain is too large (at least as large as the size required to represent the search domain). We apply backward induction to recursively solve a series of smaller games and thus overcome the size representation problem.

Secondly, we try to obtain a compact representation of the search domain. This compact representation satisfies that in every stage of the search the set of allowed queries can be computed directly from the representation in a polynomial time in the size of the search domain. In contrast, in the case of set representation, the set of allowed queries is explicitly listed in advance and is, in general, of size exponential in the size of the search domain.

Similar compact representations arise in other research areas. In particular, the problem of finding all minimum cuts in a network may be difficult, since the output may be of exponential size. In [11], a binary relation on the vertex set is defined such that a vertex subset induces a minimum cut if and only if it is a closure of the relation. In addition, in [12], it is shown that per cutset all minimum cost cutsets can be generated in polynomial time in the number of vertices in the network. In this work, for a given search domain that meets some sufficent conditions, we also introduce a binary relation and obtain a DAG representation of the search domain. This representation, analagous to the network case, enables an effective computation of the possible queries at each stage of the search. In addition, we find necessary conditions for determining if a given search domain actually represents a search in a DAG. The proof for the correctness of these conditions includes an explicit construction of the suitable DAG.

As mentioned above, the compact representation we consider in this paper is a DAG. The tree and rectangular lattice special cases are treated in [3]. In the first case, a polynomial algorithm in the size of D is obtained for finding the buggy element. In the second case, it is shown that a rectangular lattice, which is the Cartesian product of two chains of n elements, can be optimally searched in at most $2\lceil \log n \rceil$ and at least $2\lceil \log n \rceil - 2$ steps.

As search in complex domains of the above type has not been studied yet in depth, there are not many relevant works (either in Game Theory or in Computer Science). Other models of search have been considered in the context of Game Theory, e.g., nuclear reactor inspections by Michael Maschler, Shmuel Zamir and others in [9] and continuous search games [5].

In the context of computer science, search in Partial Ordered Sets of real numbers was considered by Linial and Saks [8, 7], where a query z either excludes all elements greater than z from the Poset or excludes all elements less than z. Linial and Saks proved lower and upper bounds for the number of queries needed to search in Posets in terms of some of the Poset properties. Note that in spite of the similarity in definition, their model does not fully satisfy the requirement for the complement, as a query leaves the "non-comparable" part of the Poset unchanged. Finally, the result in [3] can be viewed as a polynomial in the size of $|D| = (O(n^4 \log^3 n)$ steps) algorithm for searching in tree-like (or Forest like) Posets.

Next we consider some practical motivations for this study.

Consider the situation in which a large tree-like data structure is being transferred between two agents. Such a situation occurs when a file system (data base) is sent across a network, or a back/restore operation is done. In such cases, it is easy to verify the validity of the data in each subtree by checksum-like tests (or randomized communication complexity equality testing). Such an equality test easily detects that there is a fault, but gives no information about which node of the tree is corrupted. Using search on the tree (by querying correctness of subtrees) allows us to find the buggy node and avoid retransmitting the whole data structure.

Software testing is another motivation for studying search problems in Posets (and particularly in trees). In general, program testing can be viewed as a two person game, comprised of the tester and its 'adversary'. The adversary injects a fault into the program and the tester has to find the fault while using a minimal number of tests. A typical scenario in software testing is that the user tests the program by finding a "test bucket" (a set of inputs) that meets a certain coverage criterion, e.g., branch coverage or statement coverage [2, 4, 6]. It is plausible that in certain situations it might be possible to embed such a set of tests (e.g., the union over all test buckets that meet branch coverage) in a Poset or in a Tree, such that the requirement for covering all tests can be replaced by a requirement for searching in this Poset or Tree. Finding an optimal search can save many tests, as the cost of a search might be considerably smaller than the size of the domain. For example, the syntactic structure of a program forms a tree; thus, if suitable tests are available, statement coverage might be replaced by a search in the syntactic tree of the program.

Finally, a possible motivation and direct application is in the area of information retrieval: consider a 'Yahoo' search like scenario. Yahoo contains an immense tree that classifies home pages (currently estimated as about $1 - 2\%$ of the total number of WWW homepages). In a typical search, a node is reached and it exposes the next level of the tree (or part of it). The user chooses the appropriate branch according to the query she has in mind. But, this tree is quite deep, which often results in numerous queries before the target is reached. Clearly, such a top-down search might be inefficient compared to the optimal search of the Yahoo tree (e.g., searching in a chain of n nodes requires n queries if we execute a top-down search and only $\log n$ queries if we allow a query of arbitrary nodes). At any point in the search, such a search algorithm will allow the user to start the search in an arbitrary node other than the current root, thus minimizing the number of queries.

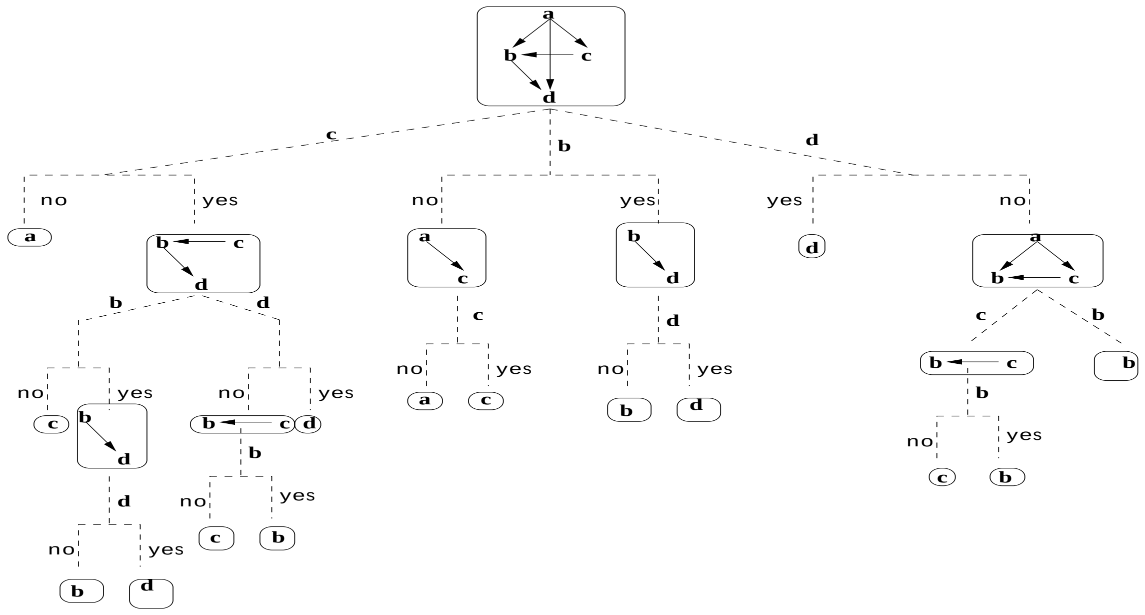


Figure 1: *Search domain of a small DAG.*

2 DAGs as search domains

In this section we consider the special case of searching in DAGs (Directed Acyclic Graphs). As explained in the introduction, one can search in DAGs directly by querying sub-graphs, thus eliminating the need for the set of sets representation, which is usually too large for practical use. We define the special case of searching in DAGs and show the equivalent representation as set of sets.

Definition 2.1 *Let $G =< V, E >$ be a DAG, $C_G(u)$ the connected component starting at node $u \in V$ and $G - C_G(u)$ its complement graph. A search in G for a buggy node $v \in V$ involves querying a node $u \in V$, and if $v \in C_G(u)$ then the search continues with $G' = C_G(u)$; otherwise, it continues with the complement $G - C_G(u)$ (until $|G| = 1$).*

The search domain in this case is usually complex, since for every stage in the search it contains not only the set of allowed queries, but also the search domain for each query ($C_G(u)$) in case the answer is 'yes', and the search domain of the complement $G - C_G(u)$ if the answer is 'no'. Figure 1 describes the search domain of a small DAG with four nodes. The sub-graphs that remain after a 'yes'/'no' answer are in oval frames, while the possible queries are marked by nodes beside the dashed lines. It follows that the best strategy is to start to query $'b'$, and that two queries are enough.

We use sets as a uniform representation for any type of search domain we may encounter. The set's members encode the different parts of the domain that can be queried at any stage. The only requirement is that the complement set of a query can be further used in the search domain in case the answer is 'no'. Figure 2 describes the same search as that of figure 1; however, it uses *sets of nodes* to encode sub-graphs and their complement graphs. Obviously, sets can be used to represent search in DAGs; however, sets can also be used to encode general types of queries which

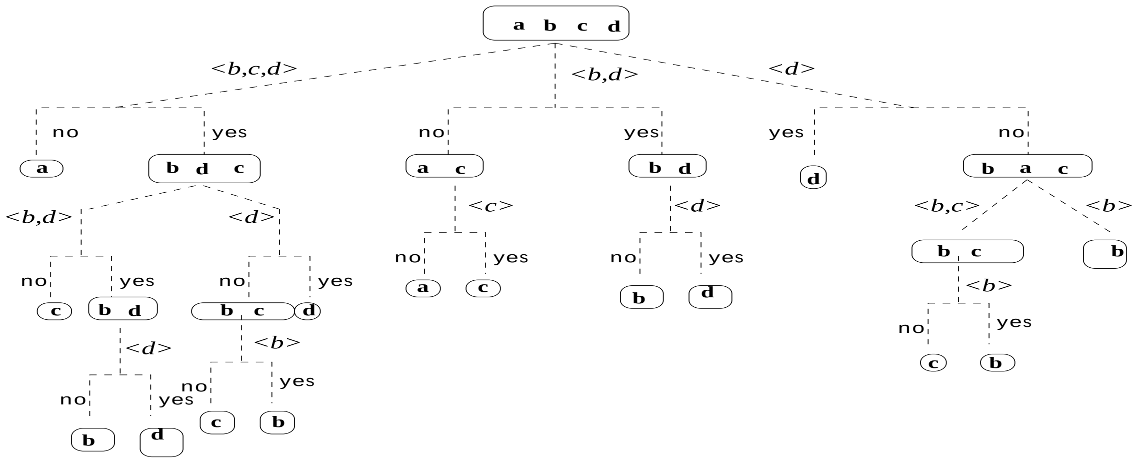


Figure 2: *Set representation of the above DAG.*

do not fit the DAG framework. In particular, sets can be used to encode specific instances of search. For example, adding the query $< a, c >$ or $< c, d >$ or both to the search of figure 1 will violate definition 2.1, yet can be added to that of figure 2 without any difficulty. Hence, using sets allows us to encode arbitrary types of queries.

3 Game definitions

Intuitively, a "searching game" is a search game on the elements of a finite set D (a search domain). A player, called "the hider" selects an element of D and marks it as "buggy". Independently, the other player, called "the searcher", tries to locate the buggy element of D. In each stage of the searching game the searcher can query a subset $D_i \subset D$ out of a pre-designated set of allowed queries $\{D_1, \ldots, D_k\}$. The set of queries is defined inductively for every D_i and its complement $D \backslash D_i$, until $|D_i| = 1$ or $|D \backslash D_i| = 1$. The query is true if D_i contains the buggy element, and false otherwise. If the query is true, the game is continued on the subset D_i; otherwise, the searcher continues the game on the complementary subset $D \backslash D_i$. The search terminates when $|D| = 1$, i.e., the buggy element has been detected. The cost of the game is the number of steps needed in order to find the buggy element. Note that we allow the hider to change the buggy element during each stage of the search, as long as the new choice is consistent with the queries made so far. As will be explained later on, this does not increase the power of the hider in the game.

The game definition includes the definition of: the search domain, the set of strategies, the game matrix and the game value.

The search domain contains the set of allowed queries and is defined as follows:

Definition 3.1 *The search domain R_D associated with a given set $D = \{d_0, \ldots d_{n-1}\}$ is a collection $R_D = \{S_{\alpha_1}, \ldots, S_{\alpha_n}\}$ of indexed* [1] *sets of queries $S_{\alpha_i} = \{\beta^i_1, \ldots \beta^i_{k_i}\}$, where $\beta^i_1, \ldots \beta^i_{k_i}$ are possible queries that can be made at α_i. S_R satisfies the following rules:*

[1] The set α_i forms the index.

- *R_D is not empty, i.e., $S_D \in R_D$ and for each $S_{\alpha_i} \in R_D$ we have that $\alpha_i \neq \emptyset$.*
- *Queries are subsets of D, i.e., $\alpha_i \subset D$ and $\beta^i_j \subset \alpha_i$.*
- *The search domain allows us to query until the buggy element has been located; i.e., for $\beta^i_j \in S_{\alpha_i}$: if $|\beta^i_j| > 1$ then $S_{\beta^i_j} \in R_D$ and if $|\alpha_i \backslash \beta^i_j| > 1$ then $S_{\alpha_i \backslash \beta^i_j} \in R_D$.*
- *If two different queries $S_{\alpha_i}, S_{\alpha_j}$ are in R_D then $\alpha_i \neq \alpha_j$.*

For example a possible search domain for $D = \{a, b, c, d\}$ may include the following queries:

$$R_D = \left\{ \begin{array}{l} S_{<a,b,c,d>} = \{< a, c >, < d >\} \ , \ S_{<a,b,c>} = \{< a, c >, < b >\} \ , \\ S_{<a,c>} = \{< c >, < a >\} \ , \ S_{<b,d>} = \{< b >\} \end{array} \right\}$$

We start the search by selecting one of the queries β^i_j in $\alpha_i \in S_D$ and if β^i_j contains the buggy element we continue with $S_{\beta^i_j}$; otherwise, we continue with the complement $S_{\alpha_i \backslash \beta^i_j}$, until we query a set of size one which evidently contains the buggy element. This process dictates the set of strategies for both players:

Definition 3.2 *Given a search domain R_D, a pure strategy of the hider is a choice function, $H : R_D \to D$ s.t. $H(S_{\alpha_i}) \in \alpha_i$. In other words, for each search domain $S_{\alpha_i} \in R_D$ the hider chooses an element $H(S_{\alpha_i})$ as the buggy element for this set of queries. A pure strategy of the searcher is a specific search in R_D, i.e., a binary tree Q_D of queries where the left subtree indicates the search in case the answer is 'yes' and the right subtree indicates the search in case of a 'no' answer:*

$$Q_{\alpha_i} = \begin{cases} \alpha_i & |\alpha_i| = 1 \\ (\beta^i_j : Q_{\beta^i_j}, Q_{\alpha_i \backslash \beta^i_j}) & otherwise\ (where \beta^i_j \in S_{\alpha_i}) \end{cases}$$

Note that by the above definition, Q_D must start with S_D. For example, a possible search strategy for the above $R_{a,b,c,d}$ may start by querying $< d >$ or $< a, c >$ as follows:

$$Q_{<a,b,c,d>} = (< d >:< d >, (< a, c >: (< a >:< a >, < c >), < b >))$$

The number of queries needed by $Q_{<a,b,c,d>}$ to find the buggy element depends on the specific strategy H_j chosen by the hider. For example, if

$$H_j(< a, b, c, d >) = H_j(< a, b, c >) = H_j(< a, c >) =' a'$$

then $Q_{<a,b,c,d>}$ requires three queries to find $'a'$.

In this game the player's strategies are not sensitive to history (such games are usually referred as a "games without perfect recall" [1] pp. 32), as for a given strategy Q_D, each set S_{α_i} can appear only once, so that the "move" of Q_D in S_{α_i} is not dependent on the history. The reason is that α_i can not belong both to a set and its complement.

Note that the sets of all possible strategies of both the hider and the searcher are finite. This suggests that a search game for a given R_D can be defined as a simple matrix game:

Definition 3.3 *Let $H_1, \ldots, H_l$ and $Q_D^1, \ldots Q_D^k$ be the set of all possible hide and search strategies of R_D. Then the search game g_D is a zero sum matrix game, such that:*

- *The columns of the game matrix M_D are indexed by the search strategies $Q_D^1, \ldots Q_D^k$, while the rows are indexed by strategies of the hider, namely $H_1, \ldots, H_l$.*
- *The payoff in the Q_D^i, H_j entry of M_D is the number of queries used by Q_D^i until a single element (the buggy element according to H_j)) is found. The game value V_g is the Nash value [10], i.e., the payoff for two strategies that are in Nash equilibrium [10].*

Note that not all zero-sum matrix games have Nash equilibrium in pure strategies, while Nash equilibrium is guaranteed for mixed strategies in zero-sum matrix games [1]. We conclude the section with the following example of a searching game which can not be solved using pure strategies.

Consider $D = \{a_1, \ldots, a_n\}$ and a searching game g_D where the searcher can query only single elements of D, i.e.:

$$R_D = \left\{ \begin{array}{l} S_{<a_1,\ldots,a_n>} = \{< a_1 >, \ldots, < a_n >\} \\ S_{<a_2,\ldots,a_n>} = \{< a_2 >, \ldots, < a_n >\} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ S_{<a_i,\ldots,a_n>} = \{< a_i >, \ldots, < a_n >\} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ S_{<a_{n-1},a_n>} = \{< a_{n-1} >, < a_n >\} \\ \textit{and so forth for every complement} \end{array} \right\}$$

Consider the mixed strategies of query/selecting each element $a_i \in S_\alpha$ with equal probability for each possible H_j or Q_D^j. These strategies are in equilibrium since both searcher and hider 'see' the same situation, i.e., all rows and columns of M_D are isomorphic. This symmetry yields that the value of V_g depends only on the size of D,

$$V_g \;=\; V_g(n) \;=\; 1 + \frac{1}{n} V_g(1) + \frac{n-1}{n} V_g(n-1).$$

As $V_g(1) = 0$, it follows that $V_g(n) = O(\frac{n}{2})$. Note that there can be no pure strategies in equilibrium, since for any choice of a_i, either the searcher or the hider can improve their payoffs.

4 Decomposing the search game

In this section we seek to find another representation of the search game, in which the "big" matrix of the game is replaced by a set of sub-matrices organized as a DAG, so that the size of the representation and the ability to compute the Nash equilibrium improve. Computing Nash-equilibrium in pure strategies (both the value and the strategies) of an $n \times m$ matrix game requires $O(n \cdot m)$ steps for finding an entry in M_D which is the maximum in its column and the minimum in its row [10]. In our case the number of strategies might be exponential in the size of the domain (or even grater) making the computation of the Nash equilibrium impractical.

We next give an example of a search game demonstrating the source of the expected savings by using a different representation. Given $D =< d_1, \ldots, d_n >$, R_d is defined using the sets

$$\alpha_i =< d_i, \ldots, d_{n-i+1} > \quad \beta_i =< d_i, \ldots, d_{n-i} > \quad \gamma_i =< d_{i+1}, \ldots, d_{n-i+1} > ,$$

such that

$$S_{\alpha_i} = \{\beta_i \ \gamma_i\} \quad S_{\beta_i} = \{\alpha_{i+1}\} \quad S_{\gamma_i} = \{\alpha_{i+1}\} \quad i = 1, \ldots, \frac{n}{2} - 1 \ .$$

The structure of R_D is therefore

$$\alpha_1 \begin{array}{c} \nearrow \beta_1 \searrow \\ \searrow \gamma_1 \nearrow \end{array} \alpha_2 \begin{array}{c} \nearrow \beta_2 \searrow \\ \searrow \gamma_2 \nearrow \end{array} \ldots\ldots \alpha_{\frac{n}{2}-2} \begin{array}{c} \nearrow \beta_{\frac{n}{2}-2} \searrow \\ \searrow \gamma_{\frac{n}{2}-2} \nearrow \end{array} \alpha_{\frac{n}{2}-1} .$$

Clearly, there are 2^n possible strategies in this search domain (following every path), so that the size of the game matrix is exponential in n. However, the total number of different queries in R_D is only $O(n)$. The expected improvement will be achieved if we are able to compute the Nash value directly on the structure of R_D, without generating the "big" matrix of the game. In this case, the Nash value can be computed using a "backward induction" on R_D, e.g., the Nash value of S_{α_i} in the above example will be computed using the Nash-value of $S_{\beta_i}, S_{\gamma_i}$, which will be computed based on the Nash-value of $S_{\alpha_{i+1}}$, and so forth. In this way the search game is decomposed into a set of of sub-matrices according to the structure of R_D. As can be seen later, this decomposition yields a game which is similar to a Game of Exhaustion, as described in [10] (pp. 89). The backward induction that we use here resembles the one used by Kuhn [1].

The actual decomposition of g_D is defined as follows:

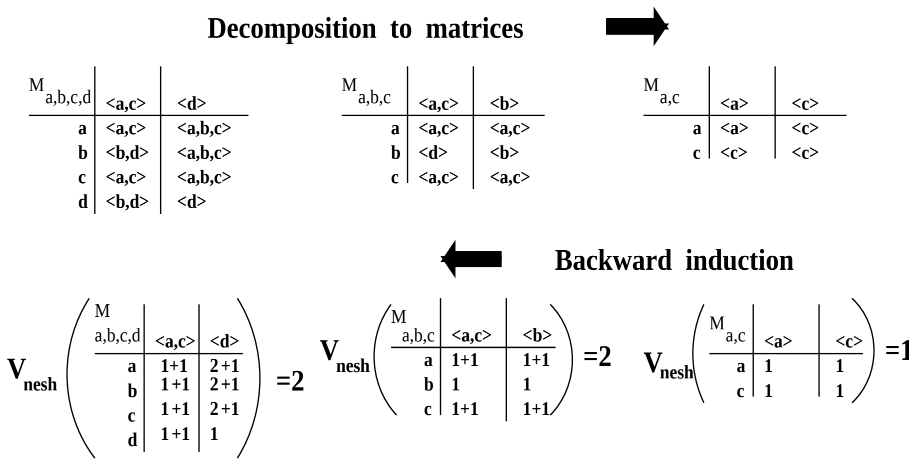
Definition 4.1 *The decomposed search game rg_D of a search domain $R_D = \{S_{\alpha_1}, \ldots, S_{\alpha_k}\}$ is a sequence of matrices $rg_D = \{M_{\alpha_1}, \ldots, M_{\alpha_k}\}$ such that:*

- *The columns of M_{α_i} are $\beta^i_1, \ldots, \beta^i_k$ and the rows are indexed by α_i's elements.*
- *Let $\alpha_i = \{d_1, \ldots, d_n\}$. The entries of each matrix are set such that:*

$$M_{\beta^i_j, d_l} = \begin{cases} \beta^i_j & d_l \in \beta^i_j \\ \alpha_i \backslash \beta^i_j & otherwise \end{cases}$$

The hider chooses an element $d_l \in D$ and the searcher chooses a search strategy, i.e., decides which β^i_j to query at each matrix Game.

The value of the game $V(rg_D)$ is defined recursively for each matrix M_α as the Nash value of the zero-sum game of M_α where all the entries $M_{\beta^i_j, d_l} = \alpha$ have been replaced by the Nash value of M_α plus one.



$M_{a,b,c,d}$	<a,c>	<d>
a	<a,c>	<a,b,c>
b	<b,d>	<a,b,c>
c	<a,c>	<a,b,c>
d	<b,d>	<d>

$M_{a,b,c}$	<a,c>	<b>
a	<a,c>	<a,c>
b	<d>	<b>
c	<a,c>	<a,c>

$M_{a,c}$	<a>	<c>
a	<a>	<c>
c	<c>	<c>

$$V_{nesh}\left(\begin{array}{c|c|c} M_{a,b,c,d} & \langle a,c\rangle & \langle d\rangle \\ \hline a & 1+1 & 2+1 \\ b & 1+1 & 2+1 \\ c & 1+1 & 2+1 \\ d & 1+1 & 1 \end{array}\right) = 2 \quad V_{nesh}\left(\begin{array}{c|c|c} M_{a,b,c} & \langle a,c\rangle & \langle b\rangle \\ \hline a & 1+1 & 1+1 \\ b & 1 & 1 \\ c & 1+1 & 1+1 \end{array}\right) = 2 \quad V_{nesh}\left(\begin{array}{c|c|c} M_{a,c} & \langle a\rangle & \langle c\rangle \\ \hline a & 1 & 1 \\ c & 1 & 1 \end{array}\right) = 1$$

Figure 3: *Example of a decomposed search game and its value.*

Definition 4.2 *The value of rg_D with respect to a search domain R_D is the Nash value of M_D and is computed recursively as follows:*

$$Vrg\left(M_{\alpha_i} = \left(\begin{array}{ccc} \dots & \beta^i_j & \dots \\ \dots & \dots & \dots \\ \alpha_i \backslash \beta^i_k & \dots & \dots \\ \dots & \dots & \dots \end{array} \right) \right) = Nash\_value \left(\begin{array}{ccc} \dots & 1 + Vrg(M_{\beta^i_j}) & \dots \\ \dots & \dots & \dots \\ 1 + Vrg(M_{\alpha_i \backslash \beta^i_k}) & \dots & \dots \\ \dots & \dots & \dots \end{array} \right)$$

This recursive process is applied for every entry in M_{α_i} which contains more than one element, i.e., $|\beta^i_j| > 1$ and $|\alpha_i \backslash \beta^i_k| > 1$. Otherwise, if $|\alpha_i| = 1$ then $Vrg(M_{\alpha_i}) = 1$. We say that the value of a search domain S_{α_i} is the Nash-value computed in the above computation for the zero-sum game associated with M_{α_i}.

Consider for example the game of the above search domain $R_{a,b,c,d}$. Figure 3 describes the matrices of $rg_{<a,b,c,d>}$ and their corresponding Nash values (below), where we have assumed that $M_{b,d} = M_{a,c}$.

It follows that the set of pure strategies and payoffs are the same in both games (g_D and gr_D). However, we still have to prove that the decomposed representation rg_D can be used to compute the Nash value of g_D. Note that computing V_{rg_D} of the search domain given at the beginning of the section can be completed in $O(n^3)$ compared to the exponential time (in n) needed to compute the Nash-value of g_D in that case.

Note that not every matrix game can be decomposed into sub-matrices so that its Nash-value can be computed using backward induction. For example, the following game has no Nash equilibrium; however, its decomposition into two sub-matrices using backward induction yield a Nash value of

4:

$$\begin{pmatrix} 3 & 5 \\ 2 & 3 \\ 5 & 4 \\ 3 & 3 \end{pmatrix} \Longrightarrow \begin{matrix} \begin{pmatrix} 3 & 5 \\ 2 & 3 \end{pmatrix} \\ \\ \begin{pmatrix} 5 & 4 \\ 3 & 3 \end{pmatrix} \end{matrix} \Longrightarrow \begin{pmatrix} 3 \\ 4 \end{pmatrix} \Longrightarrow 4$$

Based on the definition of $V(rg_D)$, (see 4.2 above), there is a Nash value associated with each search domain S_{α_i}. We assume that each such value is obtained in pure strategies. We thus associate with each search domain a pair of pure strategies (d, β^i_{j0}) which are in Nash equilibrium, where $d \in \alpha_i$ and $\beta^i_{j0} \in S_{\alpha_i}$. This defines a complete pure strategy for the hider (see 3.2 above) for the game rg_D (denoted by $\mathcal{P}$). The searcher's strategy for rg_D (denoted by $\mathcal{Q}_D$) is defined inductively starting at R_D and using the predetermined query β^i_{j0} for each search domain S_{α_i}, i.e.,

$$\mathcal{Q}_{\alpha_i} = \begin{cases} \alpha_i & |\alpha_i| = 1 \\ (\beta^i_{j0} : \mathcal{Q}_{\beta^i_{j0}}, \mathcal{Q}_{\alpha_i \setminus \beta^i_{j0}}) & otherwise\ ) \end{cases}$$

Note that these strategies are not sensitive to the history of the game, and use predetermined choices. In addition, $\mathcal{P}$ and $\mathcal{Q}_D$ are evidently pure strategies in the original game g_D; however, they are not necessary in Nash equilibrium.

To facilitate the proof of the next claim, we denote by $\mathcal{P} \downarrow \alpha_i$, the restriction of the hider strategy $\mathcal{P}$ in g_D to the search domain induced by S_{α_i}. $\mathcal{P} \downarrow \alpha_i$ is clearly a hider strategy in $g_{S_{\alpha_i}}$. Similarly, $\mathcal{Q}_D \downarrow \alpha_i$ is the search strategy we get in $g_{S_{\alpha_i}}$ by inductively starting at S_{α_i} and using the predetermined query β^i_{j0} for each search domain S_{α_j}, as was done for R_D.

Lemma 4.1 *$(\mathcal{P}, \mathcal{Q}_D)$ defined in the above paragraph is a Nash equilibrium in g_D.*

Proof: For a search domain of size 1 the claim is trivial. We assume correctness for all M_{α_i} matrix games that appear in the entries of M_D (that is, for all search domain S_{α_i} that appear in the entries of M_D). Thus, we assume by induction that for all $\alpha_i \neq D$ $(\mathcal{P} \downarrow \alpha_i, \mathcal{Q}_D \downarrow \alpha_i)$ is a Nash equilibrium in $g_{S_{\alpha_i}}$ (the sub-game associated with S_{α_i} in g_D and gr_D respectively).

By the induction assumption, if one of the players deviates from $(\mathcal{P}, \mathcal{Q}_D)$ in g_D after the first move, he will lose. Therefore, the only case to consider is a change of the first move in $\mathcal{P}$ or $\mathcal{Q}_{\mathcal{D}}$ by either player in g_D. Let H and Q be a pair of strategies for the hider and the searcher in g_D and rg_D:

- $m(H, Q)$ be the payment in g_D for playing (H, Q).
- $\alpha_{H,Q}$ is the search domain that rg_D reaches after the players have played the first move in (H, Q).

Assume that the searcher deviates from $\mathcal{Q}_D$ in the first move to Q'. Then, by the induction assumption, we get that

$$m(\mathcal{P}, Q') = 1 + m(\mathcal{P} \downarrow \alpha_{\mathcal{P},Q'}, Q' \downarrow \alpha_{\mathcal{P},Q'}) \overset{induction}{\geq} 1 + m(\mathcal{P} \downarrow \alpha_{\mathcal{P},Q'}, \mathcal{Q}_D \downarrow \alpha_{\mathcal{P},Q'}) = 1 + Vrg(M_{\alpha_{\mathcal{P},Q'}}) .$$

Since $(\mathcal{P}, \mathcal{Q}_D)$ is in Nash equilibrium in M_D, we get that

$$1 + Vrg(M_{\alpha_{\mathcal{P},Q'}}) \geq 1 + Vrg(M_{\alpha_{\mathcal{P},\mathcal{Q}_D}}) \overset{induction}{=} 1 + m(\mathcal{P} \downarrow \alpha_{\mathcal{P},\mathcal{Q}_D}, \mathcal{Q}_D \downarrow \alpha_{\mathcal{P},\mathcal{Q}_D}) \ = \ m(\mathcal{P}, \mathcal{Q}_D)\ .$$

Hence, $m(\mathcal{P}, Q') \geq m(\mathcal{P}, \mathcal{Q}_D)$, and since the same argument can be applied to a deviation of the hider, we get that any deviation from $(\mathcal{P}, \mathcal{Q}_D)$ in g_D will cause both players to lose. $\square$

Alpha-beta pruning techniques can be used to some extent to optimize the computation of the Nash equilibrium on the DAG of matrices. For example, assume that the current maximum of a column in M_{α_i} is m and we wish to compute the Nash value of the next entry $M_{\beta^i_k}$ in that column. During the evaluation of $M_{\beta^i_k}$ we find that the minimum of Nash values in some row of $M_{\beta^i_k}$ is less than m. Clearly there is no need to further evaluate the remaining Nash-values in that row, as the outcome Nash-value of $M_{\beta^i_k}$ of the current column can not exceed m.

5 Search domains versus search in graphs

In this section we consider a different type of solution to the problem of the large space required to represent search domains. Basically, we observe that in several generic cases of search domains (namely, searching in graphs or trees) there exists a more compact representation of R_D, (namely the graph or the tree itself). Consider the size of a search domain R_G corresponding to the search in a DAG (directed acyclic graph) $G =< V, E >$, where each connected component in G forms a query. If the query is answered by a 'yes' the search continues on the queried connected component; otherwise, the search continues in the connected component's complementary sub-graph. When G is known, we can use a set of nodes to identify a search domain, i.e., use R_V instead of R_G. For example, the search domain of the following graph G is as follows:

$$G = \begin{array}{ccc} a & \longrightarrow & c \\ \downarrow & & \downarrow \\ b & \longrightarrow & d \end{array} \quad \begin{array}{ll} S_{<a,b,c,d>} =< b, d >< c, d >< d > & \textit{all subgraphs in G} \\ S_{<b,d>} =< d > \quad S_{<c,d>} =< d > & \textit{theirsubgraphs} \\ S_{<a,c>} =< c > \quad S_{<a,b} =< b > & \textit{complements ofb} \longrightarrow \textit{d and c} \longrightarrow \textit{d} \\ S_{<a,b,c>} =< b >< c > & \textit{complement of d in G} \end{array} \tag{1}$$

Clearly, the size of R_V might be exponential in $|V| \ = \ n$. For example, the search domain of a rooted star (a tree with $n + 1$ nodes and n leaves) contains at least 2^n different sets[2].

Obviously, we could have used the graph itself as a compressed representation of the search domain R_G, since all the information regarding sub-graphs and their complements can be directly obtained from the graph itself. For example, we can obtain a search strategy for a given DAG $G =< V, E >$, by finding a node $u \in V$ that minimizes the difference between the size of the sub-graph rooted at u and the size of its complement (e.g., the node b in the above example). Clearly, such a node can be computed by an exhaustive search in n^2 steps, and can be used as the first query of the underlying strategy. The rest of the nodes in this strategy can be found using the same procedure on the sub-graph rooted at u and on its complement. This might not be the optimal strategy for the graph; however, it can be used as a good approximation for the optimal

[2]It is possible to show that on the average the size of the DAG's search domain is exponential in n.

strategy, if the query structure is somewhat similar to a binary query structure, e.g., the degree of G is bounded. Moreover, if G is a tree, then the algorithm proposed in [3] can be used to obtain an optimal strategy in $O(n^4 \log^3 n)$ steps, applied directly on the tree itself.

It is therefore better to represent search domains as trees or graphs, and avoid the penalty involved with the oversized general representation R_D. However, as will be shown next, not every search domain can be represented as a DAG. It is therefore important to determine whether a given search domain R_D can be so represented. In this section we find such conditions and show that they can be used to actually construct a search graph out of a given search domain that satisfies these conditions. We refer to the resulting graph as a "search graph" which is a "compressed" representation of a given search domain R_D. Formally we require that every search strategy for R_D (that satisfies the abovementioned conditions) will be a search strategy in the resulting search graph having the same payment, and vice-versa. Consequently, an optimal strategy for searching in the search graph is an optimal strategy for the original search domain.

Given a search domain R_D, let G_α denote a possible graph for the sub domain $S_\alpha \in R_D$ such that an optimal search algorithm in G_α is also an optimal strategy for the search game g_α. It is logical to assume that if $\beta \in S_\alpha$ then G_β is a sub graph of G_α. The reason is that there must be a node in G_α that corresponds to β; hence, a 'yes' answer on that node will leave us with G_β. This observation can be used to show that not every R_D can be compressed into a search graph. Consider, for example, the search domain R_D given by

$$S_{<a,b,c>} = \{< b,c >< a,b >< c >\} S_{b,c} = \{< c >\} S_{a,b} = \{< b >\}\} .$$

The only graph possible for $S_{b,c}$ is a path $b \longrightarrow c$, as the other alternatives (such as the graph with no edges) will not represent $S_{b,c} = \{< c >\}$ accurately. If we complete $G_{<a,b,c>}$ to $a \longrightarrow b \longrightarrow c$, we contradict the fact that $< a,b > \in S_{<a,b,c>}$ is a legal query. Any other completion of $b \longrightarrow c$ leads to a similar contradiction. Consequently, there is no search graph for $S_{<a,b,c>}$. We therefore seek to find necessary and sufficient conditions that determine whether or not the search in a given R_D can be compressed into a DAG. We also seek some effective construction to transform a search domain that satisfies these conditions into a DAG, so that the computation of a search strategy can be made efficient.

The proposed criterion is based on a simple observation; namely, that for every set $\beta \in S_\alpha$ there must be a **unique** node $v \in G_\alpha$ such that querying v in G is equivalent to querying β in S_α.

The discussion below is focused on search domains meeting the following two conditions.

- For every S_α there is a history (i.e., a legal search and 'hiding' sequence) that reaches S_α.
- The singletons search domain are all members of R_D, i.e., for every $d \in D$, $S_d = \{d\} \in R_D$. A search in domain problem meeting this condition is called a search in domain problem with singletons.

As there is no use in searching sub-domains that are never reached, and as every search in domain problem can be completed to a search in domain problem with singletons without affecting its value, the conditions mentioned above don't, intuitively, restrict the family of 'search in domain' problems under consideration.

In what follows we concentrate on finite acyclic connected graphs with a unique root vertex and at least two vertices. We usually refer to them simply as graphs.

For a graph $G = (V, E)$ we denote the connected component starting at $v \in G$ by $C_G(v)$. The unique root of G is denoted by r_G.

Given two graphs G_1 and G_2. we denote by $G_1 - G_2$ the graph obtained from G_1 by removing the vertices of G_2 and removing edges as required. In addition, we say that G_2 is a successor of G_1, denoted by $G_1 \rightarrow G_2$, if either

- G_2 is a connected component of G_1. I.e., $G_2 = C_{G_1}(v)$.
- or G_2 is a 'complementary' of a connected component. I.e., $G_2 = G_1 - C_{G_1}(v)$

In both cases v is not the unique root of G_1.

For a given graph G, we define a set of graphs Γ_G as follows.

- $G_0 = G$
- $G_i = \{G' \mid G_{i-1} \rightarrow G'\},\ i > 0$

$\Gamma_G = \bigcup_{i=0}^{\infty} G_i$

Lemma 5.1 *$\forall\, G' \in \Gamma_G,\ G'$ is a finite acyclic connected graph with a unique root.*

Proof: *The proof is by induction on the construction of Γ_G. If $G'' = C_{G'}(v)$ then the unique root of G'' is v. If $G'' = G' - C_{G'}(v)$, then the unique root of G'' is $r_{G'}$.*

Lemma 5.2 *Γ_G is finite.*

Proof: $G' \rightarrow G''$, hence $|V'| > |V''|$, so that after a finite number of steps $G_i = \emptyset$. In addition, G is finite, thus $\Gamma_G = \bigcup_{i=0}^{\infty} G_i$ is finite. $\square$

In what follows we refer to G and V interchangeably when the meaning is clear from the context.

Definition 5.1 *For a graph $G = (V,\ E)$, define R_G as the search domain for G as follows.*

- *The initial set for R_G is $D = V$.*
- *$\forall\, G' \in \Gamma_G$, the set of queries at G' is given by $S_{G'} = \{C_{G'}(v) \mid$ there is a path from $r_{G'}$ to v in $G'\}$.*
- *$R_G = R_D = \{\ S_{G'} \mid G' \in \Gamma_G\}$*

It is easy to see that this definition is the same as the intuitive definition of R_G given at the beginning of this section. We use the notation $u \xrightarrow{G} v$ to denote that v is a child of u in G.

Lemma 5.3 *Given a graph G, R_G is a legal search domain satisfying definition 3.1.*

Proof: Using the definition of Γ_G it follows that every condition of definition 3.1 is satisfied. For example, we have to prove that if $\beta^i_j \in S_{\alpha_i}$ and $|\alpha_i - \beta^i_j| > 1$ then $S_{\alpha_i - \beta^i_j} \in R_D$. However, $\beta^i_j \in S_{\alpha_i}$ iff $\exists\, G' \in \Gamma_G$ *s.t.* $r_{G'} \overset{G'}{\rightarrow} v$, $\beta^i_j = C_{G'}(v)$ and $S_{\alpha_i} = S_{G'}$ therefore $G'' = G' - C_{G'}(v)$ and $G' \rightarrow G''$, thus $G'' \in \Gamma_G$. If $G'' \in \Gamma_G$ then $S_{\alpha_i - \beta^i_j} = S_{G''} \in R_G$. $\square$

Next we obtain necessary conditions that any R_G satisfies.

Lemma 5.4 *R_G meets the following conditions:*

- *For all $S_{\alpha_i} \in R_G$ there is a unique element r_{α_i} (called the root vertex) such that $r_{\alpha_i} = \alpha_i - \bigcup_j \beta^i_j$.*
- *$S_{\alpha_i - \beta^i_{j_0}} = \{\beta \mid \beta = \beta^i_j - \beta^i_{j_0},\ j \neq j_0\}$. The last equality is up to an empty set ($\emptyset$).*
- *if $l \neq i$ and $\beta^l_k \in S_{\alpha_l} = S_{\beta^i_{j_0}}$ and $\beta^i_{j_0} \in S_{\alpha_i}$ then $\beta^l_k \in S_{\alpha_i}$*
- *If $\beta^i_k,\ \beta^i_j \in S_{\alpha_i}$ and $\beta^i_k \subseteq \beta^i_j$ then $\beta^i_k \in S_{\beta^i_j}$.*
- *Given the sequences $\{\alpha_i\}^n_1$ and $\{\alpha^{'}_i\}^{n-1}_1$, $n \geq 3$ such that $\alpha_{i+1} \in S_{\alpha^{'}_i}$ and $r_{\alpha_i} = r_{\alpha^{'}_i}$, $1 \leq i \leq (n-1)$, there exists a sequence $\{\beta_i\}^n_1$ such that $\beta_{i+1} \in S_{\beta_i}$, $1 \leq i \leq (n-1)$ and $r_{\alpha_i} = r_{\beta_i}$, $1 \leq i \leq n$*

Proof: Let $B = \{v \mid (r_{G'}, v) \in E'\}$ be the set of children of $r_{G'}$ in $G' = (V', E') \in \Gamma_G$ then by the construction of R_G, we get that $V' - r_{G'} = \bigcup_{v \in B} C_{G'}(v) = \bigcup_{\beta_j \in S_{G'}} \beta_j$. Thus, $S_{V'}$ (also denoted by $S_{G'}$ satisfies the first condition.

Next we show that $S_{\alpha_i - \beta^i_{j_0}} = \{\beta^i_j - \beta^i_{j_0}\}_{j \neq j_0}$. If $\beta^i_{j_0} \in S_{\alpha_i}$ then there exists a sub-graph G' such that $S_{\alpha_i} = S_{G'}$. In addition, there exists $v_0 \in V'$ where $\beta^i_{j_0} = C_{G'}(v_0)$. Furthermore, v_0 is the child of $r_{G'}$ (i.e., $r_{G'} \overset{G''}{\rightarrow} v_0$), yielding that $G'' =< V'', E'' > = G' - C_{G'}(v_0)$ where $\alpha_i - \beta^i_{j_0} = V''$. Thus,

$$S_{G''} = S_{V'' - V_{C_{G'}}(v_0)} = \{C_{G''}(v) \mid r_{G''} \overset{G''}{\rightarrow} v\}\ .$$

Using $r_{G''} = r_{G'}$ we get that $S_{G''} = \{C_{G''}(v) | \ r_{G'} \overset{G''}{\rightarrow} v\}$. The second item follows since every connected component in G'' is formed by the "subtraction" $C_{G''}(v) = C_{G'}(v) - C_{G'}(v_0)$, hence satisfying the condition $S_{\alpha_i - \beta^i_{j_0}} = \{\beta^i_j - \beta^i_{j_0}\}_{j \neq j_0}$. The third and fourth claims are trivial. Next we prove the fifth claim.

$\alpha_i,\ \alpha^{'}_i \in \Gamma_G,\ 1 \leq i \leq (n-1)$ thus α_i and $\alpha^{'}_i$ are sub-graphs of G with one root, namely $r_{\alpha_i} = r_{\alpha^{'}_i} = r_i$ and $\alpha_i, \alpha^{'}_i \subseteq C_G(r_i)$. Likewise, $r_n = r_{\alpha_n}$ and $\alpha_n \subseteq C_G(r_n)$. Let $\beta_i = C_G(r_i),\ 1 \leq i \leq n$. By the definition of β_i we have that $r_{\alpha_i} = r_{\beta_i},\ 1 \leq i \leq n$. In addition, $\alpha_{i+1} \in S_{\alpha^{'}_i},\ 1 \leq i \leq (n-1)$. Thus, by the definition of Γ_G there is a path from $r_i = r_{\alpha^{'}_i}$ to $r_{i+1} = r_{\alpha_{i+1}}$ in $\alpha^{'}_i \in \Gamma_G$ and thus in G. Therefore, $\beta_{i+1} = C_G(r_{i+1}) \subseteq \beta_i = C_G(r_i)$. In addition, $\beta_i = C_G(r_i) \in S_D$ and thus by the fourth claim above $\beta_{i+1} \in S_{\beta_i}$. $\square$

The above conditions are also sufficient to construct a search graph.

Definition 5.2 *The search graph $G = < V, E >$ of a given search domain $R_D = \{S_{\alpha_1}, \ldots, S_{\alpha_n}\}$ that satisfies the conditions of Lemma 5.4 is constructed as follows. The vertices of G are the elements of D, i.e., $V = D$. The set of edges E include all edges $(r_{\alpha_i}, r_{\alpha_l})$, such that*

- $\beta^i_k \in S_{\alpha_i}$, $S_{\beta^i_k} = S_{\alpha_l}$, *and there is no* $\beta^i_j \in S_{\alpha_i}$ *such that* $\beta^i_k \subseteq \beta^i_j$.

For example, applying this construction on $R_{a,b,c,d}$ of example 1 will reconstruct the original graph. This is because :

- $a \longrightarrow b$ and $a \longrightarrow c$ are in the graph since $r_{<a,b,c,d>} = a$, $r_{<b,d>} = b$, $r_{<c,d>} = c$ and $< b, d >, < c, d > \in S_{<a,b,c,d>}$, and there is no $\beta \in S_{<a,b,c,d>}$ that contains either $< b, d >$ or $< c, d >$.
- $b \longrightarrow d$ is in the graph since $r_{<b,d>} = b$ and $d \in S_{<b,d>}$ (similarly we obtain that $c \longrightarrow d$ is in the graph).
- There are no additional edges in the graph. For example, even though $d \in S_{<a,b,c,d>}$, there exists $\beta^i_j = < b, d > \in S_{<a,b,c,d>}$ such that $< b, d >$ contains $< d >$.

Note that every vertex $v \in G$ is named by possibly more than one r_α element (at least one is guaranteed, as we assume a search in domain problem with singletons). For a $v \in G$ denote by $R(v)$ the set of all r_αs such that $v = r_\alpha$. As we concentrate on search in domain problems with singletons, we have that $\bigcup_{v \in G} R(v) = V = D$.

We first prove the following claims:

Lemma 5.5 *Let R_D satisfy the conditions of Lemma 5.4 and G the corresponding search graph obtained by using the construction of definition 5.2. Let $G' = < V', E' > = C_G(v)$ be a connected component in G, if $v = r_{\alpha_i}$ for some $\alpha_i \in S_D$ then $V' = \alpha_i$.*

Proof: Note that it is not clear a-priori that v is the root of some set in R_D; however, as $v \in D$, then it may happen that $v = r_{\alpha_i}$. In this case, for every node $v_k \in V'$ there is a path $v_0 \longrightarrow v_1 \longrightarrow \ldots \longrightarrow v_k$, such that $v_0 = v = r_{\alpha_i}$. By the construction of definition 5.2 and the fifth condition of lemma 5.4, each $v_j = r_{\alpha_{i_j}}$ such that $\alpha_{i_j} \in S_{\alpha_{i_{j-1}}}$, $v_{j-1} = r_{\alpha_{i_{j-1}}}$ and so forth, until $v_0 = r_{\alpha_{i_0}} = r_{\alpha_i}$. The transitiveness of the third condition of lemma 5.4 implies that each $\alpha_{i_j} \in S_{\alpha_i}$, yielding that $v_k \in \alpha_i$ and $V' \subseteq \alpha_i$.

For the other direction, assume that $d \in \alpha_i$. We will then construct a path in G from r_{α_i} that ends in d, yielding that $d \in V'$. If $d = r_{\alpha_i}$ then we are done. Otherwise, let $i_0 = i$ since $r_{\alpha_{i_0}} = \alpha_{i_0} - \cup\beta^{i_0}_j$, then there exists $\alpha_{i_1} = \beta^{i_0}_{j_0}$ such that $d \in \alpha_{i_1}$ and $\beta^{i_0}_{j_0}$ is maximal (namely there is no $\beta^{i_0}_{j_1} \in S_{\alpha_{i_0}}$ such that $\beta^{i_0}_{j_0} \subset \beta^{i_0}_{j_1}$). By the construction of G there must be an edge $< r_{\alpha_{i_0}}, r_{\alpha_{i_1}} >$. It is evident that this process can be repeated until we reach a query α_{i_k} such that $\alpha_{i_k} = d$. $\square$

Another claim that is used associates a graph in $\Gamma(G)$ with every $S_{\alpha_i} \in R_D$.

Lemma 5.6 *For any given $S_{\alpha_i} \in R_D$ there is a graph $G' \in G^k$ of Γ_G such that $r_{\alpha_i} = root(G')$.*

Proof: Let $Q = \alpha_{i_0}, \alpha_{i_1}, \ldots, \alpha_{i_k}$ be a sequence of queries in R_D that reaches S_{α_i}, such that $\alpha_{i_0} \in S_D$ and $\alpha_{i_k} = \alpha_i$ and either $\alpha_{i_j} \in S_{\alpha_{i_{j-1}}}$ (if the answer is 'yes') or $\alpha_{i_j} \in S_{\alpha_{i_{j-2}} - \alpha_{i_{j-1}}}$. In addition, we chose Q to be maximal, i.e., if $\alpha_{i_j} \in S_X$ then there is no other set in S_X that contains α_{i_j}. By the construction of G and the definition of Γ_G, there is a sequence $G_0, ..., G_k$ such that $G_0 = G$, $G_j \in G^j$ and either $G_j = C_{G_{j-1}}(r_{\alpha_{i_j}})$ if the answer for α_{i_j} is 'yes' or $G_j = C_{G_{j-2}-G_{j-1}}(r_{\alpha_{i_j}})$ if the answer for α_{i_j} is 'no'. Thus, we get that $r_{\alpha_i} = root(G_k)$. □

In what follows, and when it is clear from the context, we use a connected component $C_G(u)$ to denote the set of nodes V_u of $C_G(u)$. In addition, we sometimes automatically refer to nodes as roots of queries, i.e., use r_α instead of $u, v \in G$. This is justified using the following claim:

Claim 5.1 *If $C_{G'}(u) =< \beta, E >$, $S_{C_{G'}(u)} = S_\beta$ and $S_\beta \in R_D$ then $u = r_\beta$.*

Proof:

$$r_\beta = \beta - \bigcup_{\beta_j \in S_\beta} \beta_j = \beta - \bigcup_{v \neq u} C_{G'}(v) = \beta - \bigcup_{<u,v> \in G'} C_{G'}(v) = C_{G'}(u) - \bigcup_{<u,v> \in G'} C_{G'}(v) = u$$

□

We can now show that the conditions of Lemma 5.4 are sufficient, and allow us to construct a search graph:

Theorem 5.1 *Let G be a search graph obtained by the construction of definition 5.2 applied to R_D. Let R_G be the search domain induced by G according to definition 5.1, then $R_D = R_G$.*

Proof: The proof is by induction on the construction of $\Gamma_G = \{G^0, G^1, \ldots, G^t, \ldots\}$, i.e., showing that the theorem follows if each $G' \in G^t$ satisfies a certain claim. We next explain why an induction on the construction of Γ_G covers R_G and R_D. The first set is covered, as R_G is defined by the inductive construction of Γ_G. The second set R_D is covered by this induction, as by Lemma 5.6 every S_{α_i} is "covered" by some graph in Γ_G.

The induction claim is that for any $G' =< V', E' > \in G^t$ we have that $S_{G'} = S_{\alpha_i}$, where r_{α_i} is the root of G' according to the construction of Lemma 5.6 and $S_{G'} = \{V_u |\ C_{G'}(u) =< V_u, E_u >, \ u \in G',\ u \neq root(G')\}$. Note that this also implies that $V' = \alpha_i$ as $\alpha_i = r_{\alpha_i} \bigcup \bigcup_{\beta \in S_{\alpha_i}} \beta_j$.

The induction base, $S_D = S_G$, holds as follows:

- By the construction of G and the first condition of 5.4, it is clear that $r_D = root(G)$. For any $u \in G, u \neq root(G)$ there is a path in G, $u_1 = r_D, u_2, \ldots u_k = u$ that leads to u. The construction of G (definition 5.2) and the fifth condition of lemma 5.4 yield that $u_j = r_{\alpha_{i_j}}$, such that $\alpha_{i_j} \in S_{\alpha_{i_{j-1}}}$. Using the conditions of Lemma 5.4 we get that $\alpha_{i_k} \in S_D$. By Lemma 5.5 $V_u = \alpha_{i_k}$, hence $V_u \in S_D$ and $S_G \subseteq S_D$.

- For $\beta \in S_D$ either $< r_D, r_\beta > \in G$ or by the construction of G, there exists $\beta' \in S_D$ (maximal) such that $\beta \subset \beta'$ and $< r_D, r_{\beta'} > \in G$. This process can be repeated forming a path in G from r_D to r_β. Now, using Lemma 5.5 yields that $C_G(r_\beta) = \beta$, thus $\beta \in S_G$ and $S_D \subseteq S_G$.

Assume correctness for all $G' \in G^t$ and consider $G'' \in G^{t+1}$:
First case $G'' = C_{G'}(v)$

By the induction hypothesis $C_{G'}(v) =< \beta, E >$ such that $\beta \in S_{G'} = S_{\alpha'}$ where $G' =< \alpha', E >$ and $root(G') = r_{\alpha'}$. The goal is to prove that $S_\beta = S_{C_{G'}(v)}$. Let $C_{G'}(u) =< \gamma, ... >, \gamma \in S_{C_{G'(v)}}$. By the induction claim we get that $\gamma \in S_{\alpha'}$. Clearly, $\gamma \subset \beta$ as $C_{G'}(u)$ is a connected component of $C_{G'(v)}$. Thus, the last condition of Lemma 5.4 yields that $\gamma \in S_\beta$, and $S_{C_{G'}(u)} \subseteq S_\beta$.

For the second direction, assume that $\gamma \in S_\beta$ and $\beta \in S_{\alpha'}$. By the third condition of Lemma 5.4, $\gamma \in S_{\alpha'}$. Hence, by the induction claim there is some connected component $C_{G'}(u) =< \gamma, ... >$, yielding that

$$C_{G'}(u) = \gamma \subset \beta = C_{G'}(v) \quad .$$

The fact that $C_{G'}(v)$ is a connected DAG, yields that there must be a path from v to u in G'. Thus, $\gamma = C_{G'}(u) \in S_{C_{G'}(v)}$, so that $S_\beta \subseteq S_{G''}$.

Second case $G'' = G' - C_{G'}(r_\beta)$ where $\beta \in S_{V'}$
Let $r_{\alpha'}$ be the root of G' and $u \xrightarrow{G} v$ denote that there is a path from u to v in G. By the induction hypothesis $V'' = V' - C_{G'}(r_\beta) = \alpha' - \beta$, hence

$$S_{G''} = \{C_{G''}(u) | \ r_{\alpha'} \xrightarrow{G''} u, u \in \alpha' - \beta\}$$

By claim 5.1 $u = r_\gamma$ such that $\gamma \in S_{G'}$, yielding that

$$S_{G''} = \{C_{G''}(r_\gamma) | \ r_{\alpha'} \xrightarrow{G''} r_\gamma, \ r_\gamma \in \alpha' - \beta, \ \gamma \in S_{\alpha'}\} \quad .$$

Using the fact that G' is connected we get that

$$S_{G''} = \{C_{G'}(r_\gamma) - C_{G'}(r_\beta) | \ \gamma \in S_{\alpha'}\} - \emptyset \ .$$

(Note that empty sets are generated by the intersection of $C_{G'}(r_\gamma)$ that is contained in $C_{G'}(r_\beta)$). By the second condition of Lemma 5.4 and the induction claim, we get that

$$S_{G''} = \{\gamma - \beta | \ \gamma \in S_{\alpha'}\} - \emptyset = S_{\alpha' - \beta} \quad .$$

$\square$

For a graph G a search algorithm is naturally defined as a pure strategy in R_G. Thus, if a search in domain problem meets the conditions of 5.4, an algorithm that can efficiently search graphs can be used to search for the buggy element in R_D.

References

[1] R. J. Aumann and S. Hart. *Handbook of Game Theory, Vol. 1.* Elsevier Science Publisher, 1992.

[2] Boris Beizer. *Software Testing Techniques.* Van Nostrand Reinhold, 1990.

[3] Y. Ben-Asher, E. Farchi, and I. Newman. Optimal search in trees. In *8'th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA97), New Orleans*, 1997.

[4] Phyllis G. Frankl and Elaine J. Weyeker. Provable improvements on branch testing. *IEEE Transactions on Computer*, 9(10), September 1994.

[5] S. Gal. *Continuous Search Games, Chapter 3 of Search Theory: Some Recent Developments.* Marcel Dekker. D. V. Chudnovsky and G. V. Chudnovsky (eds.).

[6] Joseph R. Horgan, Saul London, and Michael R. Lyu. Achieving software quality with testing coverage measures. *IEEE Transaction on Software Engineering*, October 1993.

[7] N. Linial and M. Saks. Every poset has a central element. *Journal of combinatorial theory*, 40:86–103, 1985.

[8] N. Linial and M. Saks. Searching ordered structures. *Journal of algorithms*, 6:86–103, 1985.

[9] M. Maschler. A price leadership method for solving the inspector's nonconstant-sum game. 1966.

[10] Guillermo Owen. *Game Theory.* Academic Press, Inc., 1982.

[11] J.C. Picard and M. Queryranne. On the structure of all minimum cuts in a network and applications. *Math. Program. Study*, 13.

[12] J. S Provan and D. R. Shier. A paradigm for listing (s, t)-cuts in graphs. *Algorithmica*, forthcoming.