Complexity analysis of heuristic CSP search algorithms

Igor Razgon

Computer Science Department, University College Cork, Ireland
i.razgon@cs.ucc.ie

Abstract. CSP search algorithms are exponential in the worst-case. A trivial upper bound on the time complexity of CSP search algorithms is $O^*(d^n)$, where n and d are the number of variables and the maximal domain size of the underlying CSP, respectively.
In this paper we show that a combination of heuristic methods of constraint solving can reduce the time complexity. In particular, we prove that the FC-CBJ algorithm combined with the fail-first variable ordering heuristic (FF) achieves time complexity of $O^*((d-1)^n)$, where n and d are the number of variables and the maximal domain size of the given CSP, respectively. Furthermore, we show that the combination is essential because neither FC-CBJ alone nor FC with FF achieve the above complexity. The proposed results are interesting because they establish connection between theoretical and practical approaches to CSP research.

1 Introduction

CSP search algorithms are exponential in the worst-case. An upper bound on the time complexity of CSP search algorithms is $O^*(d^n)$, where n and d are the number of variables and maximal domain size, respectively (we use O^* notation to suppress polynomial factors in the complexity expression). This upper bound is obtained by taking into account that a search algorithm assigns n variables and for every variable, the branching factor is at most d. Thus the $O^*(d^n)$ upper bound is not "tight". On the other hand, in the constraint satisfaction area there are many sophisticated pruning techniques. An interesting question is, whether a combination of heuristic pruning methods can reduce the time complexity.

In this paper we answer the question affirmatively. In particular, we show that the FC-CBJ algorithm [5] combined with the fail-first variable ordering heuristic (FF) [4] has the worst-case time complexity of $O^*((d-1)^n)$. Furthermore, we show that the use of both the conflict-directed backjumping and the FF are essential for reducing complexity by demonstrating that FC-CBJ with no specified ordering heuristic and FC with FF both have a complexity greater than $O^*((d-1)^n)$.

We do not claim that the combination of FC-CBJ with the FF achieve the best time complexity for constraint satisfaction problem. In fact, there are methods that solve CSP much more efficiently [3]. The main contribution of the

present paper is providing evidence that the complexity of CSP solving can be improved by using heuristic methods that were designed for the purely "practical" purpose of improving the runtime of CSP solving on real-world instances.

The results proven in this paper also provide theoretical insight into the strength of "most constrained first" ordering heuristics studied in [2]. The fact that FF improves the complexity of FC-CBJ while leaves the complexity of FC unchanged indicates that the strength of an ordering heuristic depends on the underlying search algorithm.

The rest of the paper is organized as follows. Section 2 contains the relevant background. Section 3 is the central in the paper. In this section we prove that FC-CBJ with FF has worst-case complexity of $O^*((d-1)^n)$. In Section 4 we show that neither FC with the FF nor FC-CBJ do not achieve the above complexity. Section 5 concludes the paper.

2 Preliminaries

2.1 Notations and terminology

The present paper considers only binary CSPs. A CSP Z consists of three components. The first component is a set of variables. Every variable has a domain of values. We denote a value val of a variable v by $\langle v, val\rangle$. The set of domains of variables comprises the second component of Z. The *constraint* between variables u and v is a subset of the Cartesian product of the domains of u and v. A pair of values $(\langle u, val_1\rangle, \langle v, val_2\rangle)$ is compatible if it belongs to the constraint between u and v. Otherwise the values are incompatible (*conflicting*). The set of all constraints comprises the third part of Z.

A set P of values of different variables is *consistent* (*satisfies* all the constraints) if all the values of P are mutually compatible. In this case, we call P a *partial solution* of Z. If we let $\langle u, val\rangle \in P$, we say that P *assigns* u. Accordingly, $\langle u, val\rangle$ is the *assignment* of u in P. Let V' be a subset of the set of variables assigned by P. We denote by $P(V')$ the subset of P that assigns V'. If P assigns all the variables, it is a *solution* of P. The task of a CSP search algorithm is to find a solution of Z or to report that no solution exists.

Generally, not every partial solution is a subset of a full solution. If a partial solution P is not a subset of any solution, it is called a *nogood*. Note that sometimes in the literature, the notion of nogood has a broader meaning in that it includes also a set of assignments with inner conflicts. In the present work a nogood is a specific case of a partial solution, that is, a consistent set of assignments.

2.2 The FC, FC-CBJ algorithms, and FF ordering heuristic

Forward Checking algorithm (FC) [5] is a CSP search algorithm based on enumeration of partial solutions. It starts from the empty partial solution. In every iteration, FC selects an unassigned variable, assigns it with a value and appends

to the current partial solution. The characteristic feature of FC is that whenever new assignment is added to the current partial solutions, the values of unassigned variables that are incompatible with the new assignment are temporarily removed from the domains of the variables. Therefore, when we consider some state that occurs during execution of FC, we frequently refer to the *current domain* of some variable v, having in mind the subset of values that were not removed from the domain of v.

If FC assigns all the variables of the underlying CSP, it returns a solution. However, during the iterative enlargement of the current partial solution, the current domain of some unassigned variable might be emptied. In this case, FC backtracks, that is, discards the last assignment of the current partial solution and replaces it by another assignment of the same variable. Note that when an assignment is discarded, all the values, removed because of incompatibility with the assignment, are restored in their current domains. It may also happen that FC cannot replace the discarded assignment by another one. In this case it backtracks again. Finally, it might happen that FC tries to backtrack, but the current partial solution is empty. In this case, FC reports insolubility.

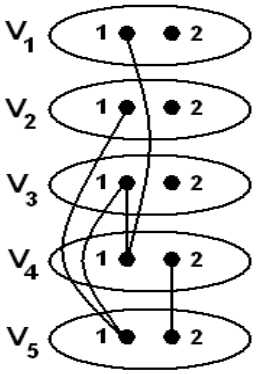

Fig. 1. CSP used for illustration of work of search algorithms

We demonstrate a possible scenario of execution of FC on the CSP shown in Figure 1, where ellipses represent variables, black circles represent values, arcs between values represent conflicts. FC starts from the empty current partial solution. Then $\langle v_1, 1\rangle$ is appended to the current partial solution and $\langle v_4, 1\rangle$ is removed because of incompatibility with $\langle v_1, 1\rangle$. The next assignment appended to the current partial solution is $\langle v_2, 1\rangle$; the assignment causes removal of $\langle v_5, 1\rangle$. The next appended assignment is $\langle v_3, 1\rangle$. Then FC adds to the current partial solution assignment $\langle v_4, 2\rangle$; as a result, $\langle v_5, 2\rangle$ is removed, the domain of v_5 is emptied and FC backtracks.

Performing backtrack, FC discards $\langle v_4, 2\rangle$ and removes it from the current domain of v_4. As well, $\langle v_5, 2\rangle$ is restored in the current domain of v_5. The back-

tracking empties the domain of v_4, hence FC backtracks again, discarding $\langle v_3, 1\rangle$ and restoring $\langle v_4, 2\rangle$. Note that $\langle v_4, 1\rangle$ is not restored because it was removed by incompatibility with $\langle v_1, 1\rangle$, which still belongs to the current partial solution Then $\langle v_3, 2\rangle$ is appended to the current partial solution. After that FC appends again $\langle v_4, 2\rangle$ which causes three consecutive backtracks discarding assignments $\langle v_4, 2\rangle$, $\langle v_3, 2\rangle$, and $\langle v_2, 1\rangle$. Then FC appends $\langle v_2, 2\rangle$ to the current partial solution. The assignment $\langle v_3, 1\rangle$ appended next is discarded after a number of iterations. After appending of assignments $\langle v_3, 2\rangle$, $\langle v_4, 1\rangle$ and $\langle v_5, 1\rangle$, FC obtains a full solution, which is returned.

States of a search algorithm The execution of a CSP search algorithm can be represented as a sequence of atomic operations of updating of the current partial solution (addition or removal of assignments) accompanied by appropriate updating of the maintained data structures in order to preserve consistency. The information recorded in the data structures before the beginning of the execution or after performing an atomic operation constitutes a *state* of a search algorithm. Thus a sequence of states is another possible representation of a search algorithm. We use this representation in the present paper, in order to prove properties of the analyzed algorithms.

Forward Checking with Conflict-directed Backjumping (FC-CBJ) is a modification of FC that can backtrack more than 1 step backwards (backjump). The completeness of enumeration is preserved by maintaining *conflict sets* of variables. In a given state of FC-CBJ, the conflict set of a variable v, denoted by $conf(v)$, contains all variables whose assignments in the current partial solution are "culprit" for removing values from the current domain of v. In particular, if P is the current partial solution then every removed value $\langle v, val\rangle$ of v is incompatible with $P(conf(v))$ or $P(conf(v)) \cup \{\langle v, val\rangle\}$ is a nogood.

The detailed description of FC-CBJ is quite technical and long, hence we list only those features of the algorithm that are relevant to the theorems we are going to prove.

- Initially all conflict sets are empty.
- Whenever a value $\langle u, val\rangle$ is appended to the current partial solution and a value of an unassigned variable v is removed as a result of incompatibility with $\langle u, val\rangle$, u is added to $conf(v)$.
- Whenever the empty domain of a variable v causes backtrack, FC-CBJ backjumps to the last assigned variable u that appears in $conf(v)$ and discards the assignment of this variable. The assignments of variables that were appended to the current partial solution after the assignment of u are just canceled as if they were not performed at all (of course, with the restoring of values removed by these assignments). Note that removing an assignment of a variable from the current partial solution, FC-CBJ removes appearances of this variable from all conflict sets.
- Whenever the empty current domain of a variable v causes backtrack and the backtrack process discards the assignment of a variable u, $conf(u)$ is set to $conf(u) \cup conf(v) \setminus \{u\}$.

Assume that the CSP illustrated on Figure 1 is processed by FC-CBJ. In the beginning, the execution is similar to that of FC with the only difference that whenever new assignments are appended to the current partial solution the conflict sets of the corresponding variables are updated. In particular adding assignment $\langle v_1, 1\rangle$ causes adding v_1 to $conf(v_4)$, v_2 is added to $conf(v_5)$ as a result of appending of $\langle v_2, 1\rangle$. Note that the assignment $\langle v_3, 1\rangle$ does not cause updating of conflict sets. Finally the assignment $\langle v_4, 5\rangle$ causes adding v_4 to $conf(v_5)$. The first backtrack of FC-CBJ is caused by the empty domain of v_5. At the time of the backtrack, $conf(v_5) = \{v_2, v_4\}$, hence FC-CBJ jumps to v_4. Note that before the backtrack, $conf(v_4) = \{v_1\}$. After backtrack the set is updated to $\{v_1, v_2\}$ as a result of union with $conf(v_5)$ and removing of v_4. Also, v_4 is removed from $conf(v_5)$.

The second backtrack occurs because of emptying of the domain of v_4. Because the last variable in $conf(v_4)$ is v_2, FC-CBJ jumps over v_3 and discards the assignment of v_2. Thus FC-CBJ avoids processing of an unnecessary assignment $\langle v_3, 2\rangle$ performed by FC after the second backtrack.

CSP search algorithms do not specify explicitly the order of selection of variables to be assigned. This job is done by ordering heuristics. One of the simplest and the most successful ordering heuristics is called Fail-First (FF) [4]. Every time when a new variable must be assigned, FF selects a variable with the smallest size of the current domain. The time complexity of FF is linear in the number of unassigned variables. The implementation of FF requires maintaining array of domain sizes of variables which is updated dynamically when values are removed or restored. All what FF does is selection of the minimal element among the entries of the array that correspond to the domains of unassigned variables.

Consider the execution of FC with FF on the CSP of Figure 1, assuming that in case of existence of two or more variables with the smallest domain size, one is selected according to the lexicographic ordering.

Initially, the current domains of all the variables are of equal size, so v_1 is assigned with 1. After removing of $\langle v_4, 1\rangle$ as a result of the assignment, v_4 becomes the variable with the smallest domain size, so $\langle v_4, 2\rangle$, the only remaining value is appended to the current partial solution. The value $\langle v_5, 2\rangle$ is removed because of the incompatibility with $\langle v_4, 2\rangle$, hence v_5 becomes the variable with the smallest domain size and the assignment $\langle v_5, 1\rangle$ is added to the current partial solution. The values $\langle v_2, 1\rangle$ and $\langle v_3, 1\rangle$ are incompatible with $\langle v_5, 1\rangle$, hence they are removed from the current domain of v_2 and v_3. The next two iterations append to the current partial solution $\langle v_2, 2\rangle$ and $\langle v_3, 2\rangle$. The obtained full solution is returned after that.

The above example demonstrates the strength of FF, because it allows to avoid backtracks during processing of the given CSP.

2.3 Complexity of backtrack algorithms

All complete CSP search algorithms (those that return a solution if one exists or report insolubility otherwise) have exponential time-complexity. Discussing as-

pects related to the complexity of backtracking algorithms, we follow two agreements:

- We express the time-complexity (upper bound) by O^* notation [7], which suppresses the polynomial factor. For example, instead of $O(n^2 * 2^n)$, we write $O^*(2^n)$. Note, that for a constant $d > 1$ $O^*((d-1)^n)$ is smaller than $O^*(d^n)$ because $O^*(d^n) = O^*((d/d-1)^n * (d-1)^n)$, where $(d/(d-1))^n$ is a growing exponential function that cannot be suppressed by the O^* notation. On the other hand, given constants d and k, $O^*(d^{n+k})$ is the same as $O^*(d^n)$ because $O^*(d^{n+k}) = O^*(d^k * d^n)$, where d^k can be suppressed as a constant.
- We express the time complexity of a CSP search algorithm by the number of partial solutions generated by the algorithm. This is a valid representation because the time complexity can be represented as the number of partial solutions multiplied by a polynomial factor which is ignored by the O^* notation.

The worst-case complexity of FC and FC-CBJ when applied to a CSP with n variables and maximum domain size d is widely considered to be $O^*(d^n)$.

The Ω^*-notation is used to express the lower bound on the complexity of exponential algorithms. The constant and polynomial factors are suppressed analogously to the O^*-notation.

3 FC-CBJ combined with FF has $O^*((d-1)^n)$ complexity

In this section we will show that the use of heuristic techniques can decrease the complexity of a search algorithm. In particular we prove that the FC-CBJ algorithm [5] combined with the FF heuristic [4] has a worst-case complexity of $O^*((d-1)^n)$, where n and d are the number of variables and the maximal domain size, respectively. During the proof, we extensively use the notion of maximal partial solution.

Definition 1. *Let P be a partial solution explored by a search algorithm during solving a CSP Z. Then P is maximal if it is not a subset of any other partial solution visited by the algorithm during solving Z.*

We now prove a theorem that states an upper bound on the number of maximal solutions explored by FC-CBJ with FF. The overall complexity of FC-CBJ with FF will follow from this result.

Theorem 1. *FC-CBJ with FF applied to a CSP Z with $n \geq 2$ variables and maximal domain size d explores at most $M(n) = d * \Sigma_{i=0}^{n-2}(d-1)^i$ maximal partial solutions.*

In order to prove the theorem, we need an additional lemma.

Lemma 1. *Let Z be a CSP with the maximal domain size d. Consider a state S of FC-CBJ that occurs during processing of Z. Let P be the current partial*

solution maintained by FC-CBJ in this state. Assume that in P is not empty and that the current domain size of every unassigned variable is d. Let Z' be a CSP created by the current domains of unassigned variables. Assume that Z' is insoluble. Then, after visiting state S, the execution of FC-CBJ is continued as follows: FC-CBJ detects insolubility of Z', immediately discards all the values of P, reports insolubility, and stops.

Proof. Considering that d is the maximum possible domain size, we infer that the current domains of the unassigned variables are the same as their initial domains. It follows that all values of the original domains of the unassigned variables are compatible with all values of P. Consequently, the conflict sets of all the unassigned variables are empty.

Observe that when processing Z', a variable assigned by P does not appear in any conflict set of a variable of Z'. This observation can be verified by induction on the sequence of events updating the conflict sets of Z'. Note that the observation holds *before* FC-CBJ starts to process Z', because all the conflict sets are empty (see the argumentation in the previous paragraph). Assuming that the observation holds for the first k events, let us consider the $k+1$-th one. Assume that v is the variable whose conflict set is updated. If this updating results in insertion of the currently assigned variable then the variable being inserted belongs to Z' which is not assigned by P. Otherwise, $conf(v)$ is united with the conflict set of another variable u of Z'. However, $conf(u)$ does not contain variables assigned by P by the induction assumption.

If Z' is insoluble, FC-CBJ will eventually discard P. This means that FC-CBJ will arrive at a state in which the current domain of a variable v of Z' is empty and $conf(v)$ does not contain any variable of Z'. On the other hand, $conf(v)$ will not contain any variable assigned by P. That is, the conflict set of v will be empty. Consequently, FC-CBJ will jump "over" all the assigned variables, report insolubility of Z, and stop. ■

Now we are ready to prove Theorem 1.

Proof of Theorem 1. We prove the theorem by induction on n. For the basic case assume that $n = 2$. Let v_1 and v_2 be the variables of Z and assume that v_1 is assigned first. Consider the situation that occurs when v_1 is assigned with a value $\langle v_1, val\rangle$. If the value is compatible with at least one value in the domain of v_2, FC-CBJ returns a solution. Otherwise, it instantiates $\langle v_1, val\rangle$ with another value of v_1 or reports insolubility if all values of v_1 have been explored. Thus, every value of v_1 participates in at most one partial solution. Keeping in mind that there are at most d such values, we get that at most d partial solutions are explored. Observe that $M(2) = d$. That is, the theorem holds for $n = 2$.

Assume that $n > 2$ and that the theorem holds for all CSPs having less than n variables. We consider two possible scenarios of execution of FC-CBJ.

According to the first scenario whenever the current partial solution is not empty (at least one variable has been already instantiated), FC-CBJ combined with FF selects for instantiation a variable with the current domain size smaller than d. Then FC-CBJ explores a search tree in which at most d edges leave the root node and at most $d-1$ edges leave any other node.

Note that when FC-CBJ has assigned all the variables but one, it does not execute branching on the last variable. If the domain of the last variable is not empty, FC-CBJ takes any available value and returns a full solution. Otherwise, it backtracks. It follows that in the search tree explored by FC-CBJ only the first $n-1$ levels can contain nodes with two or more leaving edges. The branching factor on the first level is d, but the branching factor of a node at any other of $n-2$ remaining levels is $d-1$. Consequently, the number of leaves of the search tree is at most $d*(d-1)^{n-2}$. Taking into account that the leaves of the search tree correspond to the maximal partial solutions, we see that in the considered case, FC-CBJ explores at most $d*(d-1)^{n-2} \leq M(n)$ maximal partial solutions. Thus, the theorem holds in the case of the first scenario.

If the first scenario does not occur then FC-CBJ, having at least one variable instantiated, selects for assignment a variable with the current domain size d. Consider the first time when such a selection occurs and denote by P the current partial solution maintained by FC-CBJ in the considered state. Denote by Z' the CSP created by the current domains of variables that are no assigned by P. Proceeding the execution, FC-CBJ solves Z'. If Z' is soluble then FC-CBJ finds a solution of Z', returns its union with P, and stops.

The case when Z' is insoluble is **the main point in the proof of the theorem**. Note that FC-CBJ uses FF. If a variable with the current domain size d is selected, the current domain sizes of the other unassigned variables are at least d. On the other hand, d is the maximal possible domain size, hence the current domain sizes of the other variables are exactly d. By Lemma 1, FC-CBJ stops after detecting insolubility of Z'. (Note that both FC-CBJ and FF contributed to the validity of this claim. The contribution of FF is ensuring that the current domains of all the unassigned variables are exactly d. The contribution of FC-CBJ is explained in the proof of Lemma 1. Note also that Lemma 1 has been proven for the general case of FC-CBJ, hence it holds, in particular, for FC-CBJ combined with FF.)

Thus we have shown that whenever FC-CBJ selects a variable with the current domain size d given that the current partial solution is non-empty, the algorithm always stops when the solving of Z' is finished.

The number of maximal partial solutions visited by FC-CBJ in this case equals the number of maximal partial solutions explored before visiting P plus the number of maximal partial solution explored after visiting P.

Recall that we consider the first time during the execution when a variable with the current domain size d is selected given that the current partial solution is not empty. Therefore, before arriving to the considered state, FC-CBJ explores at most $d*(d-1)^{n-2}$ maximal partial solutions, according to the argumentation provided for the first scenario.

All maximal partial solutions explored after visiting P are visited during solving of Z'. Therefore every maximal partial solution P_1 visited after exploring of P can be represented as $P_1 = P \cup P_2$, where P_2 is a maximal partial solution of Z' (non-maximality of P_2 contradicts maximality of P_1). Thus the number

of maximal partial solutions explored after visiting of P equals the number of maximal partial solutions explored by FC-CBJ during solving of Z'.

Considering that P is not empty, it follows that Z' contains at most $n-1$ variables. By the induction assumption, FC-CBJ explores at most $M(n-1)$ of maximal partial solutions during solving of Z'. Thus the overall number of maximal partial solutions is at most $d*(d-1)^{n-2}+M(n-1)=M(n)$, what completes the proof for the second scenario. ■

Corollary 1. *FC-CBJ with FF explores $O^*((d-1)^n)$ maximal partial solutions.*

Proof. By definition of $M(n)$, $M(n) \leq dn(d-1)^{n-2} = O^*((d-1)^n)$. ■

We have shown that the number of maximal partial solutions explored by FC-CBJ with FF is bounded by $O^*((d-1)^n)$. Clearly, every partial solution is a subset of some maximal partial solution. On the other hand, every maximal partial solution serves as a subset of at most n partial solutions. Indeed, every partial solution generated by FC-CBJ corresponds to a node of the search tree explored by FC-CBJ. Note that subsets of the given partial solution P correspond to the ancestors of P in the search tree. Taking into account that every path from the root to a leaf in the search tree has a length of at most n, we infer that P cannot have more than n ancestors. Consequently, the number of partial solutions explored by FC-CBJ is at most the number of maximal partial solutions multiplied by n. Thus we have proved the following theorem.

Theorem 2. *The complexity of FC-CBJ with the fail-first ordering heuristic is $O^*((d-1)^n)$.*

To understand the strength of the theorem, consider the following corollary.

Corollary 2. *FC-CBJ with FF efficiently solves any CSP with at most two values in every domain.*

Proof. If a CSP contains at most two values in every domain then $d=2$. Substituting $d=2$ to the statement of Theorem 2, we get that FC-CBJ with FF solves such a CSP in $O^*(1^n)$ with is a polynomial according to the definition of O^* notation. ■

The collection of CSPs with at most 2 values in every domain is a well-known polynomially-solvable CSP class. According to Corollary 2, FC-CBJ recognizes CSPs from this class without any additional "domain-dependent" procedures.

4 Both FC with FF and FC-CBJ have a complexity greater than $O^*((d-1)^n)$

It may seem that a combination of FC-CBJ with FF is far too complex to achieve the purpose of reducing complexity. We will show that this is not so. In particular, we will show that both FC (without CBJ) with FF and FC-CBJ alone have a complexity greater than $O^*((d-1)^n)$.

Let us start by the analyzing of the complexity of FC with FF. We prove that for every n there is a CSP Z with $n+1$ variables and d values in every domain (d is an arbitrary number) such that in order to solve Z, the algorithm generates at least d^n partial solutions. Let $v_1, \ldots, v_{n+1}$ be the variables of the considered CSP. Every value of v_n conflicts with every value of v_{n+1}. There are no other conflicts in the CSP. This CSP is illustrated in Figure 2.

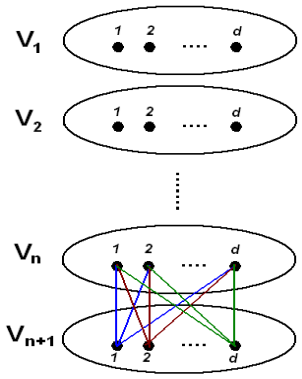

Fig. 2. A hard CSP for FC with FF

We assume that if during the search there are two or more variables with the smallest current domain size, FC with FF will select the first of them according to lexicographic ordering. This is a valid assumption, because we are going to refute the claim that FC with FF has a better complexity than $O^*(d^n)$. It is implied by the claim that if there are two or more variables with the smallest current domain size, these variables can be ordered arbitrarily. Therefore, to refute the claim, it is sufficient to show that FC combined with FF and a *particular* ordering in the case of existence of two or more variables with the smallest domain size has a complexity greater than $O^*((d-1)^n)$.

Observe that the source of insolubility of Z is that v_n and v_{n+1} have no pair of compatible values. However, FC with the heuristic described above will not be able to recognize the insolubility source, because it will assign first v_1, then v_2, and so on. Note that to refute Z, the algorithm will have to explore all the partial solutions assigning variables $v_1, \ldots, v_n$. Clearly, there are at least d^n such partial solutions. Denoting $n+1$ by m we obtain that for every m there is a CSP with m variables such that FC with FF explores at least d^{m-1} partial solutions solving this CSP. That is, the complexity of FC with FF is $\Omega^*(d^{m-1}) = \Omega^*(1/d * d^m) = \Omega^*(d^m)$ as claimed.

Let us now analyze the complexity of FC-CBJ. Note that if some CSP search algorithm has a complexity of $O^*((d-1)^n)$ then for *any* value of d, the algorithm explores $O^*((d-1)^n)$ partial solutions. Consequently, to prove that an algorithm has a greater complexity, it is enough to show that the above does not happen

for at least one d. This is the way we show that FC-CBJ alone has a greater complexity than $O^*((d-1)^n)$. Note that we are free to choose an ordering heuristic for FC-CBJ because FC-CBJ does not specify any particular ordering heuristic.

Let Z be a CSP with $n+1$ variables $v_1, \ldots v_{n+1}$. The domain of each variable contains n values, say, $val_1, \ldots, val_n$. The only conflicts of Z are found between the values of v_{n+1} and the values of other variables. In particular, a value $\langle v_{n+1}, val_i \rangle$ conflicts with all the values of variable v_i. The CSP is illustrated in Figure 3. We assume that FC-CBJ orders variables lexicographically.

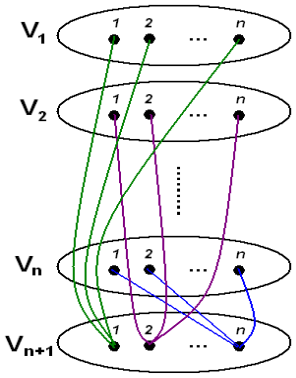

Fig. 3. A hard CSP for FC-CBJ

The source of insolubility of Z is that every value of v_{n+1} conflicts with the domain of some variable from $v_1, \ldots, v_n$. However, FC-CBJ is unable to recognize the source of insolubility because it assigns v_{n+1} last, according to the specified ordering heuristic. Observe that all maximal partial solutions generated by FC-CBJ are of the form $\{\langle v_1, val_{i_1}, \rangle, \ldots, \langle v_n, val_{i_n} \rangle\}$, because no assignment to a proper subset of $\{v_1, \ldots, v_n\}$ can discard all the values of v_{n+1}. Clearly, there are n^n partial solutions of the above form and we shall show that FC-CBJ explores all of them, which proves that for any given n, there is d for which there is a class of CSPs that cannot be solved by FC-CBJ in $O^*((d-1)^n)$.

To show that FC-CBJ explores all the partial solutions of the form $\{\langle v_1, val_{i_1}, \rangle, \ldots, \langle v_n, val_{i_n} \rangle\}$, it is sufficient to show that FC-CBJ never backjumps more than 1 step backwards when applied to Z. First, we show that FC-CBJ never backjumps when explores a maximal partial solution. Actually, an assignment of every v_i conflicts only with $\langle v_{n+1}, val_i \rangle$. That is, every assignment of a maximal partial solution conflicts with the unique value of v_{n+1}. This means that when the current domain of v_{n+1} is emptied, all the variables of $\{v_1, \ldots, v_n\}$ appear in $conf(v_{n+1})$. Therefore, after discarding the assignment of v_n, the set $\{v_1, \ldots, v_n\} \setminus \{v_n\}$ is added to the conflict set of v_n. Further, when the domain of v_n is emptied, FC-CBJ has no choice but to backtrack to v_{n-1}.

Tracing further the execution of FC-CBJ, we observe that whenever an assignment of v_i is discarded, the set $\{v_1, \ldots, v_{i-1}\}$ is added to $conf(v_i)$. Hence, when the current domain of v_i is emptied, FC-CBJ backtracks to v_{i-1}. This argumentation shows that FC-CBJ applied to Z with lexicographic ordering heuristic never backjumps and thus explores at least n^n partial solutions.

5 Conclusion

In this paper we presented complexity analysis of a few heuristic algorithm for solving of CSPs. In particular, we proved that FC-CBJ combined with FF has time-complexity of $O^*((d-1)^n)$. We have also demonstrated that the above combination of techniques is necessary is order to reduce complexity. In particular, we have proven that FC with FF as well as FC-CBJ without an ordering heuristic both have a complexity greater than $O^*((d-1)^n)$.

The results presented can be further generalized. Note that the only property of FF used in the proof of Theorem 1 is that FF *does not* select a variable with the largest current domain. Consequently, the result of Theorem 1 can be generalized for a combination of FC-CBJ with any heuristic that has the above property. Note also that FC-CBJ can be replaced by another intelligent backtracking algorithm like MAC-CBJ [6] or CCFC- [1].

References

1. Fahiem Bacchus. Extending forward checking. In *Principles and Practice of Constraint Programming*, pages 35–51, 2000.
2. C. Beck, P. Prosser, and R. Wallace. Trying again to fail-first. In *CSCLP 2004*, pages 41–55, 2004.
3. D. Eppstein. Improved algorithms for 3-coloring, 3-edge coloring and constraint satisfaction. In *SODA-2001*, pages 329–337, 2001.
4. R. M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
5. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.
6. P. Prosser. MAC-CBJ: maintaining arc consistency with conflict-directed backjumping. Technical Report Research Report/95/177, Dept. of Computer Science, University of Strathclyde, 1995.
7. G. Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization: "Eureka, you shrink", LNCS 2570*, pages 185–207, 2003.