Proc. of International Conference on Principles and Practice of Constraint Programming 1997 (Lecture Notes in Computer Science 1330, Springer), pp. 356–370

Why Adding More Constraints Makes a Problem Easier for Hill-climbing Algorithms: Analyzing Landscapes of CSPs

Makoto Yokoo

NTT Communication Science Laboratories
2-2 Hikaridai, Seika-cho
Soraku-gun, Kyoto 619-02 Japan
e-mail: yokoo@cslab.kecl.ntt.co.jp

Abstract. It is well known that constraint satisfaction problems (CSPs) in the phase transition region are most difficult for complete search algorithms. On the other hand, for incomplete hill-climbing algorithms, problems in the phase transition region are more difficult than problems beyond the phase transition region, i.e., more constrained problems. This result seems somewhat unnatural since these more constrained problems have fewer solutions than the phase transition problems.
In this paper, we clarify the cause of this paradoxical phenomenon by exhaustively analyzing the state-space landscape of CSPs, which is formed by the evaluation values of states. The analyses show that by adding more constraints, while the number of solutions decreases, the number of local-minima also decreases, thus the number of states that are reachable to solutions increases. Furthermore, the analyses clarify that the decrease in local-minima is caused by a set of interconnected local-minima (*basin*) being divided into smaller regions by adding more constraints.

1 Introduction

Recently, rapid advances have been made in research to clarify what kinds of constraint satisfaction problems (CSPs) are most difficult for complete search algorithms. These works show that the problems in the region where the solvable probability is about 50% (phase transition region) tend to be most difficult [1, 5, 7].

The above results are intuitively natural. If the problem is weakly constrained, a complete algorithm will easily find a solution. Also, if the problem is very strongly constrained, the problem becomes easy since the complete algorithm can prune most of the branches in the search tree. Therefore, the most difficult problems will exist in the middle, i.e., the phase transition region.

On the other hand, what kind of problems are most difficult for incomplete hill-climbing algorithms (such as GSAT [10]) that do not perform an exhaustive search? Since these algorithms cannot discover the fact that a problem is unsolvable, trying to solve unsolvable problems by these algorithms is futile. Therefore, if we choose solvable problem instances and apply the algorithms to these problem instances, what kinds of problems are most difficult?

At first glance, we may think that a problem becomes more difficult for hill-climbing algorithms if we add more constraints, which provides the problem with fewer solutions. However, by actually solving problems using hill-climbing algorithms, the problems in the phase transition region are the most difficult, as with the complete algorithms, although these problems have more solutions that the problems beyond the phase transition region, i.e., more constrained problems. Such paradoxical results have been reported in [2].

In this paper, we clarify the cause of this paradoxical phenomenon by exhaustively analyzing the state-space landscape of CSPs. A state is one possible assignment to all variable values, and the evaluation value of the state is the number of its constraint violations. The landscape of the state-space is formed by the evaluation values of the states. Such analyses are important not only for clarifying the cause of this paradoxical phenomenon, but also for developing more efficient hill-climbing algorithms by utilizing the results of landscape analyses.

For path-planning problems, such landscape analyses have been reported in [6]. On the other hand, as far as the author knows, there has been no research done on analyzing the landscapes of CSPs, with the notable exception of [4]. In this research, the relation between the landscapes of graph-coloring problems and the number of possible colors is analyzed.

In this paper, we first divide the states into two classes: states that are *reachable* to solutions, and states that are unreachable to solutions since they lead to local-minima. Then, we show that the problems beyond the phase transition region actually have more solution-reachable states than the problems in the phase transition region. Next, we show that the number of local-minima decreases by adding more constraints, thus the number of solution-reachable states increases. Furthermore, we examine the number and widths of *basins*, each of which is a set of interconnected local-minima. We show that the number of local-minima decreases because a basin is divided into smaller regions by adding more constraints.

In the following sections of this paper, we first describe CSPs and the algorithm used for the analyses (Section 2), then we show the analyses of the state-spaces of 3-SAT problems (Section 3). Furthermore, we discuss the obtained results in more detail (Section 4).

2 Problem Definition and Algorithm

2.1 Problem Definition

A constraint satisfaction problem can be defined as follows. There exist n variables $x_1, x_2, ..., x_n$, each of which takes its value from a finite, discrete domain $D_1, D_2, ..., D_n$, respectively. There also exists a set of constraints. A constraint among k variables $c(x_{i1}, \ldots, x_{ik})$ is a subset of a product $D_{i1} \times \ldots \times D_{ik}$, which represents allowed combinations of variable values. The goal is to find an assignment of all variables that satisfies all constraints.

A satisfiability problem for propositional formulas in conjunctive normal form (SAT) is an important subclass of CSP, and can be defined as follows. A boolean *variable* x_i is a variable that takes the value true or false. We call the value assignment of one variable a *literal*. A *clause* is a disjunction of literals, e.g., $x_1 \vee \overline{x_2} \vee x_4$. Given a set of clauses $C_1, C_2, \ldots, C_m$ and variables $x_1, x_2, \ldots, x_n$, the satisfiability problem is to determine if the formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

is satisfiable. That is, is there an assignment of values to the variables that makes the above formula true.

A 3-SAT problem is a SAT problem in which the number of literals in each clause is restricted to 3. The number of clauses divided by the number of variables is called the *clause density*, and the difficulty of a randomly generated 3-SAT problem is mainly determined by this clause density [1, 7].

2.2 Hill-climbing Algorithm

In a hill-climbing algorithm for solving a CSP, each variable has a tentative initial value, and the value of a variable is changed one by one so that the number of constraint violations decreases.

We introduce the following terms for explaining our algorithm and the analyses in the next section.

state: We call one possible assignment of all variables a *state*. In a SAT problem with n variables, the number of states is 2^n.

evaluation value: For a state s, we call the number of constraint violations of the state the *evaluation value* of the state (represented as $eval(s)$).

neighbor: For a state s, we say another state s' is a *neighbor* of s, if s' is obtained by changing one variable value of s. In a SAT problem with n variables, each state has n neighbors.

local-minimum: For a state s that is not a solution, if the evaluation values of all of its neighbors are larger than or equal to s's evaluation value, we say s is a *local-minimum*. Specifically, the following condition is satisfied: $\forall s'$ if s' is a neighbor of s, then $eval(s') \geq eval(s)$.

strict local-minimum: For a state s that is not a solution, if the evaluation values of all of its neighbors are larger than s's evaluation value, we say s is a *strict* local-minimum. Specifically, the following condition is satisfied: $\forall s'$ if s' is a neighbor of s, then $eval(s') > eval(s)$.

non-strict local-minimum: If a state s is a local-minimum, but not a strict local-minimum, we say s is a *non-strict* local-minimum. If s is a non-strict local-minimum, there exists at least one neighbor that has the same evaluation value.

In this paper, we use the algorithm described in Figure 1 for analyses. This algorithm is a greedy, unfair deterministic tie-breaking algorithm [3]. As the basic GSAT, this algorithm moves from the current state to the neighboring

state, which has the smallest evaluation value. Also, if the current state is a non-strict local-minimum, this algorithm can move to the neighboring state that has the same evaluation value (sideway move).

On the other hand, unlike GSAT (in which ties are broken randomly), if there exist multiple states that have the best (smallest) evaluation value, ties are broken in a fixed, deterministic, unfair way (e.g., each state has a unique identifier, and a tie is broken by using the alphabetical order of these identifiers). We use this method in order to simplify the analyses in the next section[1].

```
procedure hill_climbing
    restart: set s to a randomly selected initial state;
    for j:=1 to Max-flips
        if eval(s) = 0 then return s;
        else if s is a strict local-minimum then goto restart;
            else s:= a neighbor of s that has the smallest evaluation value;
            (ties are broken deterministically)
            end if;
        end if;
    end;
    goto restart;
```

Fig. 1. Hill-climbing Algorithm

In the following, we mainly use randomly generated 3-SAT problems for analyses. In a randomly generated 3-SAT problem, each clause is generated by randomly selecting three variables, and each of the variables is given the value true or false with a 50% probability. In Figure 2, we show the ratio of solvable problems, in 100 randomly generated 3-SAT problem instances with 24 variables, by varying the clause density (number of clauses/number of variables). We can see that phase transition occurs[2] when the clause density is around 4.67.

Furthermore, we select 50 solvable problem instances from randomly generated problems for each clause density, and solve these instances with the algorithm described in Figure 1. We show the average number of restarts required to solve the problem instances in Figure 3 (for each problem instance, 1000 trials are performed using different initial states). Also, we show the average number of solutions to these solvable problem instances in Figure 4.

From these figures, we can see that the average number of solutions at the clause density 4.67 (where the phase transition occurs) is 11.5, and the average number of solutions at the clause density 5.83 (beyond the phase transition)

[1] We discuss the effect of tie-breaking methods on algorithm efficiency in Section 4.2.

[2] Since we use very small-scale problems, the phase transition is not as drastic as with large-scale problems.

is 3.6 (less than 1/3 of the solutions at the phase transition region). However, the required number of restarts at the clause density 5.83 is 6.9 (about 50% fewer than for the phase transition region, i.e., 13.0). Consequently, the problems become easier.

Although we use very small-scale problem instances, we can see the paradoxical phenomenon that the problems with fewer solutions are actually easier than the problems with more solutions.

3 Analyzing State-Space

3.1 Analyzing Solution-Reachability

For each state, the neighboring state that the algorithm can move to is uniquely determined, since ties are broken in a fixed, deterministic method. The state-space of the problem can be described using a graph, in which a state is represented as a node, and a possible transition between states is represented as a directed link (Figure 5). In Figure 5, a symbol near a node represents the identifier of the state, and a value within a node represents the evaluation value of the state. Also, a dotted link connects two neighboring states that do not have a directed link between them. For example, in Figure 5, e and h are neighbors, but there is no directed link between them, since the algorithm moves to f from these states. In this example, we assume that ties are broken in the alphabetical order of identifiers.

Each node has exactly one outgoing link, except for a strict local-minimum (e.g., state g) and a solution (e.g., state f). Also, there can be a cycle that circulates around multiple non-strict local-minima (e.g., links between a and b).

By tracing directed links from a state s, the result will be either to reach a solution or not (i.e., to reach a strict local-minimum, or to go around non-strict local-minima). In the former case, we say s is *reachable* to solutions, and in the latter case, we say s is *unreachable* to solutions. For example, in Figure 5, c, e, f and h are reachable and a, b, d and g are unreachable to solutions.

In Figure 6, we show the average ratio of the states that are solution-reachable. By adding more constrains, the average ratio of solution-reachable states actually increases beyond the phase transition region.

If the probability that a randomly selected state is solution-reachable is given by p, the estimated number of restarts required to find a solution will be $1/p$ (i.e., $p + 2p^2 + 3p^3 \ldots$). In Figure 7, we show the estimated number of restarts[3] derived from Figure 6. The values in Figure 7 are almost identical to the actual measurements in Figure 3.

[3] Note that the average ratio at clause density 4.67 is 0.12, but this does not mean the estimated number of restarts will be 8.33=1/0.12. In general, the average of '$1/p$' is not equal to the reciprocal of the average of p. For example, the average of 1/5 and 1/2 is 7/20, while the reciprocal of the average of 5 and 2 is 2/7.

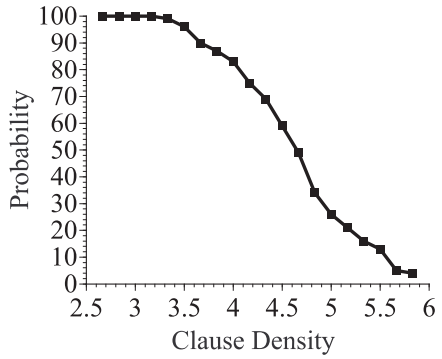

Fig. 2. Probability of satisfiability (3-SAT problems)

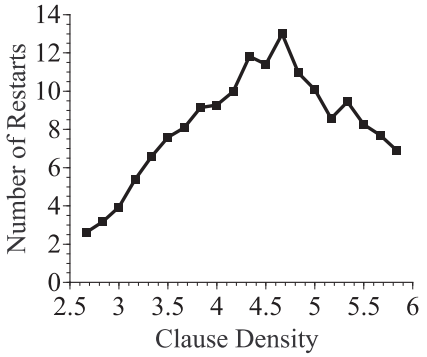

Fig. 3. Average number of restarts (3-SAT problems)

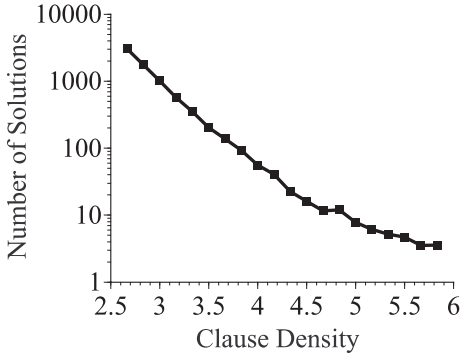

Fig. 4. Average number of solutions (3-SAT problems)

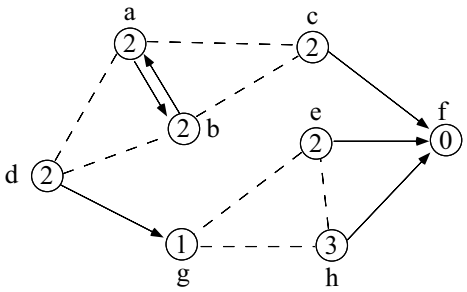

Fig. 5. Example of state-space

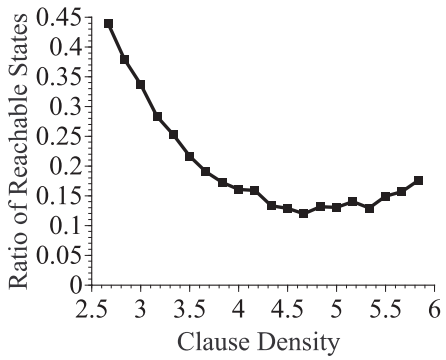

Fig. 6. Average ratio of solution-reachable states (3-SAT problems)

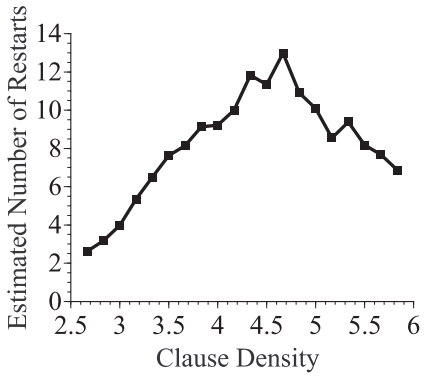

Fig. 7. Average estimated number of restarts (3-SAT problems)

3.2 Analyzing local-minima

In view of the above, why dose adding more constraints increase the number of solution-reachable states beyond the phase transition, although it decreases the number of solutions? As shown in Figure 5, tracing directed links from a state will either result in reaching a solution or reaching a local-minimum (strict or non-strict). Let us examine how the number of local-minima changes by adding constraints. Figure 8 shows the average number of local-minima. We can see that the number of local-minima decreases monotonically by adding constraints.

While adding constraints makes a problem harder for a hill-climbing algorithm by decreasing the number of solutions, it also makes the problem *easier* by decreasing the number of local-minima. As Figure 4 shows, the number of solutions decreases very rapidly around the phase transition region, and then decreases slowly beyond the phase transition region (note that the y-axis of Figure 4 is log-scaled). On the other hand, the number of local minima decreases at a more constant rate (note that the y-axis of Figure 8 is linear). Therefore, we can assume that by adding more constraints, the effect of making the problem easier will exceed the effect of making the problem harder beyond the phase transition region.

Another possible factor is that the evaluation values of local-minima will increase by adding more constraints (while the evaluation values of solutions are always 0), and thus the number of solution-reachable states will increase.

3.3 Analyzing Basins

Considering the above, why does the number of local-minima decrease monotonically by adding more constraints? Adding constraints does not necessarily mean that the number of local-minima must decrease. By adding more constraints and increasing the evaluation values of states, some of these states may change to non local-minima, but some states that are neighbors of the increased states may become local-minima.

This question can be answered by analyzing the landscape of the state-space. By checking each local-minimum, we found that almost all local-minima are non-strict local-minima (there are only a few strict local-minima for each problem) and many non-strict local-minima are interconnected (by neighborhood relation) to create a flat surface in the state-space. We call such parts *basins*.

The formal definition of a basin is as follows.

Definition: A basin is a set of connected[4] states that satisfies the following conditions.

- All states in the basin have the same evaluation value, which is larger than 0.
- For each state s in the basin, each neighboring state s' of s, which is not a member of the basin, has an evaluation value greater than or equal to the evaluation value of s, i.e., $eval(s') \geq eval(s)$.

[4] We mean *connected* by neighborhood relation in general, not only by directed links.

We say a basin is *maximal* if the above conditions are violated by adding any neighboring state. Also, we call the number of states in a basin the *width* of the basin. For example, in Figure 5, there are two maximal basins, i.e., $\{a, b\}$ and $\{g\}$. From this definition, any state in a basin is a local-minimum, any local-minimum is a member of some maximal basin, and a state cannot be a member of two different maximal basins. Therefore, the summation of widths of maximal basins is equal to the number of local-minima. Also, a strict local-minimum forms a basin that has only one member.

Figure 9 shows the average width of maximal basins. We can see that the average width of basins decreases monotonically by adding constraints. Figure 10 shows the average number of maximal basins. We can see that the number of basins increases initially, then reaches a peak, and finally slowly decreases. When a problem is very weakly constrained, adding more constraints divides a basin into several regions. When the problem is strongly constrained, each basin is small and the basins tend to be eliminated by adding more constraints. It is obvious that if a basin is divided into several regions, the summation of widths of these parts would be smaller than the original width.

To summarize, most local-minima are interconnected with each other to create basins. By adding constraints, a basin is divided into smaller regions and the summation of widths of maximal basins (the number of local-minima) decreases.

4 Discussion

4.1 3-coloring Problem

The analyses in the previous section give us a clear explanation of the paradoxical phenomenon. However, how general are these results?

To reconfirm the results in other classes of CSPs, we show the evaluation results in another important class of CSPs, i.e., graph-coloring problems. In Figure 11, we show the ratio of solvable problems in 100 randomly generated problem instances having 12 variable and 3 possible colors (3-coloring problems) by varying the link density (number of links/number of variables). We can see that phase transition occurs[5] when the link density is around 1.9.

Furthermore, we select 50 solvable problem instances from randomly generated problems for each link density and solve these instances. We show the average number of restarts required to solve the problem instances in Figure 12, and the average number of solutions of these solvable problem instances in Figure 13. As with the 3-SAT problem, we can see the paradoxical phenomenon in which the problems with fewer solutions are easier than the problems with more solutions.

Figure 14 shows the average ratio of the solution-reachable states. By adding more constrains, the average ratio of solution-reachable states increases beyond the phase transition region. Figure 15 shows the average number of local-minima.

[5] As in the case of the 3-SAT problems, the phase transition is not as drastic as with large-scale problems.

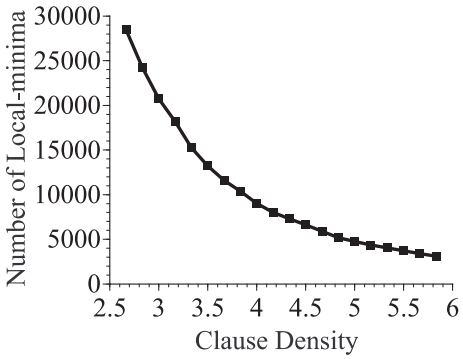

Fig. 8. Average number of local-minima (3-SAT problems)

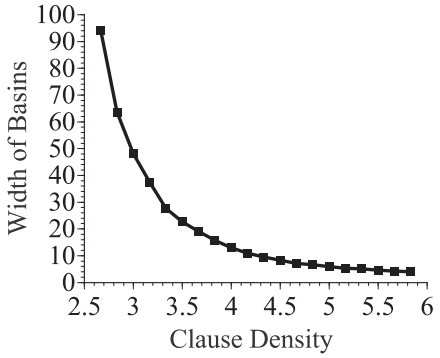

Fig. 9. Average width of basins (3-SAT problems)

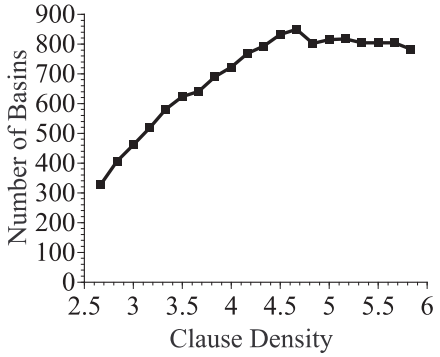

Fig. 10. Average number of basins (3-SAT problems)

As with the 3-SAT problems, the number of local-minima decreases monotonically by adding constraints, except for the part where the link density is very low. Figure 16 shows the average width of maximal basins, and Figure 17 shows the average number of maximal basins. The results are very similar to those obtained for the 3-SAT problems, except that the number of basins increases monotonically.

4.2 Effects of Tie-breaking

We use an algorithm with a unfair deterministic tie-breaking method. As discussed in [3], using an unfair tie-breaking method degrades the performance of the algorithm. In the example problems used in this paper, we can obtain about a two-fold speedup by using a random tie-breaking method.

However, a random tie-breaking method complicates the solution-reachability analyses described in Section 3.1. This is because the same state can be either solution-reachable or unreachable, depending on the tie-breaking. For example, in Figure 5, let us assume that the current state is a. If ties are broken in a fixed way as described in the figure, the algorithm circulates between a and b. On the other hand, if ties are broken randomly, the algorithm may fall into the strict local-minimum g or reach a solution via c.

However, it should be noted that the analyses in Section 3.2 and Section 3.3 are not affected by tie-breaking methods.

4.3 Problem Size

One drawback of our results is that only very small-scale problems were analyzed. In order to perform an analysis of solution-reachability or an analysis of basins within a reasonable amount of time, we need to explicitly construct a state-space within the physical memories of a computer. For example, if the number of variables of a 3-SAT problem is 30, the total number of states becomes 10^9. Even if we could manage to represent each state with 4 bytes, required memory size would be 4GB. Evaluating the number of local-minima could be done without explicitly constructing a state-space. However, we still need to exhaustively check each state. In the case of a 3-SAT problem with 40 variables, the total number of states is 10^{12}. If we could check 10^6 states per second, we would still need more than a week to analyze one problem instance.

However, although the analyses were done only for very small-scale problems, the observed phenomenon is very similar to that of larger-scale problems reported in [2]. Also, very similar results were obtained for two different classes of problems (3-SAT and 3-coloring). Therefore, we can assume that the obtained results would also be valid for large-scale problems.

5 Conclusions

In this paper, we clarified the cause of the paradoxical phenomenon that problems with fewer solutions are easier than problems with more solutions for hill-

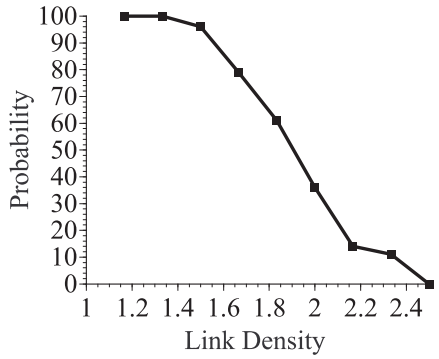

Fig. 11. Probability of satisfiability (3-coloring problems)

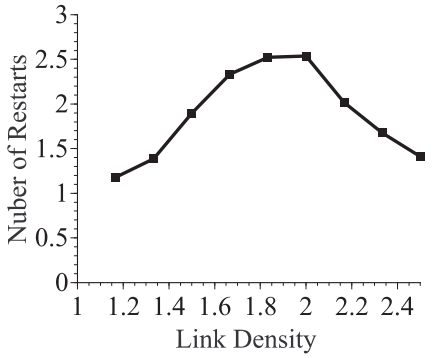

Fig. 12. Average number of restarts (3-coloring problems)

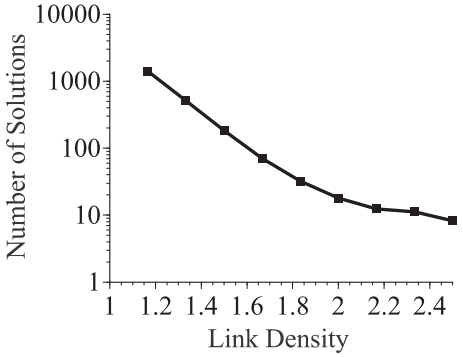

Fig. 13. Average number of solutions (3-coloring problems)

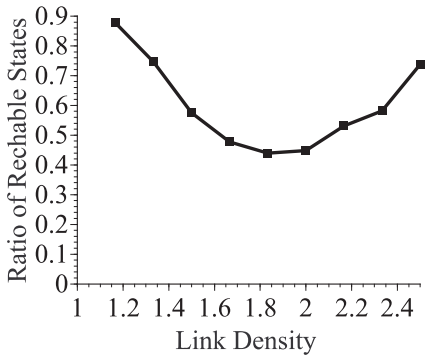

Fig. 14. Average ratio of solution-reachable states (3-coloring problems)

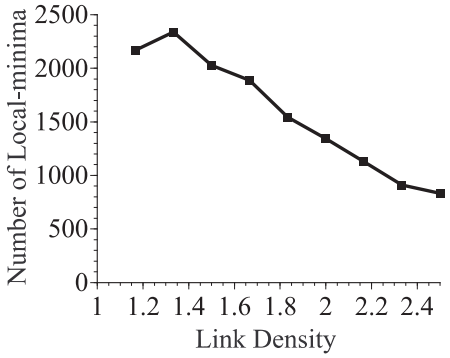

Fig. 15. Average number of local-minima (3-coloring problems)

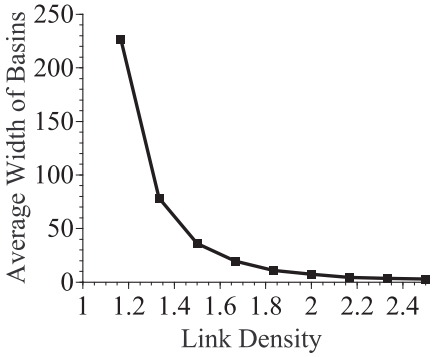

Fig. 16. Average width of basins (3-coloring problems)

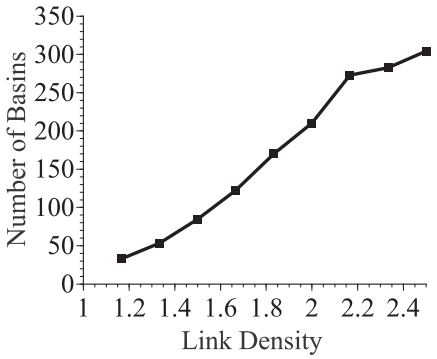

Fig. 17. Average number of basins (3-coloring problems)

climbing algorithms. This was done by exhaustively analyzing the landscape of the state-space.

The analyses of problem instances of 3-SAT problems and 3-coloring problems showed that by adding more constraints, while the number of solutions decreases, the number of local-minima also decreases, and thus the number of solution-reachable states increases. Furthermore, we showed that the number of local-minima decreases since the basins are divided into smaller regions by adding more constraints.

Our future works include analyzing how the landscape of the state-space affects different types of hill-climbing algorithms (e.g., breakout [8], walk-SAT [9]) and developing more efficient hill-climbing algorithms by utilizing the results of landscape analyses.

Acknowledgments

The author wish to thank K. Matsuda and F. Hattori for supporting this research, and K. Ye for helping the analyses.

References

1. Cheeseman, P., Kanefsky, B., and Taylor, W.: Where the really hard problems are, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (1991) 331–337
2. Clark, D. A., Frank, J., Gent, I. P., MacIntyre, E., Tomv, N., and Walsh, T.: Local Search and the Number of Solutions, *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming (Lecture Notes in Computer Science 1118)*, Springer-Verlag (1996) 119–133
3. Gent, I. P. and Walsh, T.: Towards an Understanding of Hill-climbing Procedures for SAT, *Proceedings of the Eleventh National Conference on Artificial Intelligence* (1993) 28–33

4. Hertz, A., Jaumard, B., and de Aragao, M. P.: Local optima topology for the k-coloring problem, *Discrete Applied Mathematics*, Vol. 49, (1994) 257–280
5. Hogg, T., Huberman, B. A., and Williams, C. P.: Phase transitions and the search problem, *Artificial Intelligence*, Vol. 81, No. 1–2, (1996) 1–16
6. Ishida, T.: Real-Time Bidirectional Search: Coordinated Problem Solving in Uncertain Situations, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 6, (1996) 617–628
7. Mitchell, D., Selman, B., and Levesque, H.: Hard and Easy Distributions of SAT Problem, *Proceedings of the Tenth National Conference on Artificial Intelligence* (1992) 459–465
8. Morris, P.: The Breakout Method for Escaping From Local Minima, *Proceedings of the Eleventh National Conference on Artificial Intelligence* (1993) 40–45
9. Selman, B. and Kautz, H.: Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (1993) 290–295
10. Selman, B., Levesque, H., and Mitchell, D.: A New Method for Solving Hard Satisfiability Problems, *Proceedings of the Tenth National Conference on Artificial Intelligence* (1992) 440–446