A FUZZY CONSTRAINT SATISFACTION APPROACH TO ACHIEVING STABILITY IN DYNAMIC CONSTRAINT SATISFACTION PROBLEMS

BY

WONG, YIN PONG ANTHONY

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF PHILOSOPHY IN

COMPUTER SCIENCE AND ENGINEERING

THE CHINESE UNIVERSITY OF HONG KONG

JULY 2001

A FUZZY CONSTRAINT SATISFACTION APPROACH TO ACHIEVING STABILITY IN DYNAMIC CONSTRAINT SATISFACTION PROBLEMS

SUBMITTED BY WONG, YIN PONG ANTHONY
FOR THE DEGREE OF MASTER OF PHILOSOPHY
AT THE CHINESE UNIVERSITY OF HONG KONG IN JULY 2001

ABSTRACT

Dynamic Constraint Satisfaction Problem (DCSP) is a class of problems that extends the traditional Constraint Satisfaction Problem (CSP) framework for dynamic environments, in which the sets of constraints are modified. In many real-life applications such as dynamic scheduling, after the problem is modified by external factors, it is desirable to find a new solution which is the most similar to the previous one. In the literature this is called a stability problem. As the definition of similarity between two solutions are different from problems to problems, a metric called distance is used as an objective measurement.

We propose to model the stability problem as a fuzzy CSP (FCSP) by imposing supplementary fuzzy constraints into the DCSP, with adequate assignment of degrees of satisfaction according to the distance requirement of the problem: higher degrees of satisfaction are assigned to tuples that are associated with more stable solutions, and lower degrees of satisfaction to tuples in less stable solutions. User's preference can then be easily represented by the supplementary fuzzy constraints, which serves to guide the search to regions of higher stability.

Fuzzy GENET, a local search solver, is used as our basis for building a specialized solver to not only make use of the degrees of satisfaction of the supplementary fuzzy constraints to achieve better stability, but also successively improve the best solution at hand over time. Experiments on a wide range of randomly generated problems show that solutions obtained by using our approach are very close to the optimal ones. As fuzzy GENET is not a complete

search algorithm, optimal solutions are not guaranteed. However, it is substantially faster than branch-and-bound complete search, which is advantageous in dynamic environments as quick response is often required.

利用模糊約束方法使動態約束滿足問題之解趨向穩定

黃彥邦

摘 要

動態約束滿足問題 (DCSP) 是爲了使到傳統的約束滿足問題能應用於動態環境而延伸出來的一個範疇。它的特點是問題中的約束關係有機會被變更。當問題受到外在因素所影響，以致它本身要作出改變時，我們都希望新的答案會跟舊的那個最接近。在很多現實生活的問題就是如此，例如動態調度。在文獻中，含有這個要求的問題被稱爲穩定問題 (stability problems)。然而，每個問題對於“接近”的定義都不一樣，因此我們採用距離這個單位來做客觀量度的準則。

我們提議把一些模糊約束條件附加到動態約束滿足問題中，將穩定問題化成模糊約束滿足問題來解決。這些模糊約束條件中的元組之隸屬度必需要與距離要求吻合：較穩定的解中出現的元組得到較高的隸屬度；相反，出現在較不穩定的解中的元組將會得到較低的隸屬度。根據這個模式，用者對穩定解的偏好就能輕易地利用這些附加模糊約束條件表達出來。而且，它們更能幫助引領搜索過程到含有高穩定解的區域。

基於模糊 GENET 這個局部搜索求解器，我們設計了一個可連續不斷地搜索出比已知的解更佳的求解器。在一系列隨機產生的問題之實驗中，證實用我們的方法所取得的解與最優解很接近。因爲模糊 GENET 求解器並非採用完全搜索，因此它不能保證一定能夠找到最優解，但是在運算速度上，它更比 branch-and-bound 完全搜索快很多。這個優點在動態環境下很重要，因爲這類問題一般都需要快速的回應。

Acknowledgements

It comes a really very long way to finish this thesis. Thank very much my supervisors, Jimmy Lee and H. F. Leung, for they have provided guidance during my research and spent lots of their time and effort on it. Although the reviewing cycles on the transcripts seem to be neverending, as there are always words and paragraphs to be refined, without them such a high quality piece of work would not have been possible.

Thanks to everyone who have given input and feedback to my research. As seniors, Kenneth Choi and Jason Wong (courtesy to his fuzzy GENET) helped me and taught me a lot during the early stages of my MPhil study, introducing me many things about CSP. Thanks Yousef Kilani for many fruitful discussions with him. My brother, who shares with me his MPhil study experience. And our department, without the computing resources I won't be able to produce such a complete set of experimental results.

Also thanks to all of my friends. Although nearly all of them do not understand what I'm working on and query me why it takes so much time, and one of them even tease me that I will not be able to finish it! (but it so happens that he's one of my best friends), I still remember many of theirs words of encouragement (and disencouragement). I really appreciate them.

Finally, to my family — for your patience, care, and support.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Constraint satisfaction provides a convenient and powerful framework to formulate many kinds of artificial intelligence tasks, which are often NP-hard problems. They include resource allocation, scheduling and time-tabling problems. In the traditional constraint satisfaction framework, dynamic problems, whose problem formulations change when they are being solved, are not considered. There are two aspects of such dynamic problems that are most interesting to us. One is to find a solution of the new problem efficiently, and the other is to find a solution of the new problem that is most similar to the old one. The latter belongs to the class *stability problems*, which is what we focus in this thesis. Our definition of stability follows the one proposed by Gérard Verfaillie and Thomas Schiex in [40], namely the similarity of successive solutions.

To solve the stability problem, we model it as a *fuzzy CSP* [29, 21, 33] (FCSP) by imposing additional fuzzy constraints into the altered CSP, and solve this FCSP instead of the altered one. The purpose of these fuzzy constraints is to represent the user's preference on what tuples the stable solutions are constituted of. Higher degrees of satisfaction are assigned to the fuzzy constraints' tuples that exist in more stable solutions, and lower degrees of satisfaction are assigned to the tuples that exist in less stable solutions. If the fuzzy constraints are properly defined according to the stability requirement of the problem, these

supplementary fuzzy constraints can guide the search to find more stable solutions.

We adapt fuzzy GENET [49, 50] to our need for solving the FCSP produced by using our modelling scheme. Fuzzy GENET is a local search algorithm that is fast in finding solutions. As fuzzy GENET is not a complete search algorithm, optimal solutions are not guaranteed. However, in some situations it worths sacrificing stability for efficiency. We also propose enhanced versions of fuzzy GENET that are specifically good for achieving higher solution stability.

1.1 Constraint Satisfaction Problems

A *Constraint satisfaction Problem* (CSP) is defined as a problem of finding an assignment of values to a set of variables such that they satisfy a set of relations simultaneously. We call these relations *constraints* because the values that the variables can take are constrained by them.

Here, we define a constraint satisfaction problem as:

Definition 1.1.1 (Constraint Satisfaction Problem) A constraint satisfaction problem (CSP) can be expressed as a triple (Z, D, C), where

- Z is a finite set of variables $\{z_1, z_2, \ldots, z_n\}$.
- D is a set of finite *domains*, each element D_i of which is the set of possible values for z_i.
- C is a finite (possibly empty) set of constraints, each on an arbitrary subset of Z.

Constraints in C restrict the values that the variables can take simultaneously.

A variable assignment is of the form $\langle z_{i_1}, v_{i_1} \rangle \cdots \langle z_{i_m}, v_{i_m} \rangle$, meaning that $v_{i_k} \in D_{i_k}$ is assigned to z_{i_k}, where $\{z_{i_1}, \ldots, z_{i_m}\} \subseteq Z$. When the context is

clear, we abuse notation by using the value tuple $(v_{i_1}, \ldots, v_{i_m})$ to denote also the same variable assignment. To solve a CSP, we need to find an assignment of values for all variables in Z from their corresponding domains such that all the constraints in C are satisfied simultaneously. Such an assignment is called a *solution* of the CSP.

CSP algorithms can be divided broadly into two types. One is systematic tree search [20], and the other is stochastic local search. The former builds a complete solution by choosing a value for a variable one at a time. This kind of method always maintains a consistent assignment for a subset of the variables. Consistency algorithms [20] are often applied to speed up the search. These algorithms are used to prune away domain values and relation tuples which will lead to inconsistency, so as to reduce the search space.

On the other hand, local search methods generate a possibly inconsistent assignment first, and then "repair" the variables by modifying their values until a complete solution is found. Candidates of this kind include the basic hill-climbing method [37], min-conflict heuristic [23], GSAT [34], and GENET [8]. Local search strategies usually have the disadvantage of getting stuck in "local minima," that means that the algorithm cannot find a move which would give rise to a better state, but at the same time, the current state is *not* a solution. As a result, various strategies, such as the breakout method [26], have been proposed in order to escape from or to avoid being trapped in local minima.

1.2 Solution Stability in Dynamic Constraint Satisfaction Problems

In this section we first define what a dynamic constraint satisfaction problem is. Based on it, we define the stability problem, and describe the axioms that all valid distance functions, which are metrics for measuring stability, must obey.

Definition 1.2.1 (Dynamic Constraint Satisfaction Problem) A *dynamic constraint satisfaction problem* (DCSP) is a sequence of CSPs $(P_0, P_1, \ldots, P_i, \ldots)$. Each CSP P_i, $i > 0$, is resulted from a change in the preceding one P_{i-1}. A change can either be a restriction (addition of constraints) or a relaxation (retraction of constraints) [11]. In this thesis, we only consider restrictions because solutions remain valid after a relaxation.

Definition 1.2.2 (Stability Problem) Given a tuple $(\boldsymbol{s}^i, P_i, P_{i+1}, d)$ where $\boldsymbol{s}^i$ is a solution to CSP P_i, and $P_{i+1} = P_i \cup C'$ for some set of constraints C', a *stability problem* is to find a solution $\boldsymbol{s}^{i+1}$ for P_{i+1} so that $d(\boldsymbol{s}^i, \boldsymbol{s}^{i+1})$ is minimized for the distance metric d.

Intuitively, stability implies similarity between $\boldsymbol{s}^i$ and $\boldsymbol{s}^{i+1}$. Quantitatively, distance is used as the metric for measurement. The shorter the distance, the more stable the new solution is, and *vice versa*. It is calculated by a *distance function* $d(\boldsymbol{v_1}, \boldsymbol{v_2})$, which takes two vectors of variable assignments $\boldsymbol{v_1} = (v_{1_1}, \ldots, v_{1_n})$ and $\boldsymbol{v_2} = (v_{2_1}, \ldots, v_{2_n})$ as input, and outputs the *distance*, which is a real number greater than or equal to zero. In other words, d defines a mapping $(D_{z_1} \times \ldots \times D_{z_n}) \times (D_{z_1} \times \ldots \times D_{z_n}) \rightarrow \{0\} \cup \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of positive real numbers. Moreover, given any variable assignment $\boldsymbol{v_3}$, any distance function d must meet the following metric axioms:

$$d(\boldsymbol{v_1}, \boldsymbol{v_2}) = 0 \textit{ iff } \boldsymbol{v_1} = \boldsymbol{v_2} \tag{1.1}$$

$$d(\boldsymbol{v_1}, \boldsymbol{v_2}) = d(\boldsymbol{v_2}, \boldsymbol{v_1}) \tag{1.2}$$

$$d(\boldsymbol{v_1}, \boldsymbol{v_2}) \leq d(\boldsymbol{v_1}, \boldsymbol{v_3}) + d(\boldsymbol{v_3}, \boldsymbol{v_2}) \tag{1.3}$$

Note that the above axioms automatically imply that $d(\boldsymbol{v_1}, \boldsymbol{v_2}) \geq 0$.

Since the meaning of stability varies from problems to problems, each problem has its own way to compute the distance. Users are obliged to define distance functions for their problems. Although there does not exist a distance function that fits all stability problems, examples are shown in Section 4.1 to illustrate how distance functions can be defined in different scenarios.

1.3 Motivation of the Research

The need to modify a CSP during the problem solving stage often arises in real-life applications, whether it is due to input of new information or unanticipated incidents, such as machine break-down and under- or over-estimation of resources usage. Since changes usually imply re-allocation of resources and informing the parties involved, it is ideal to find a new solution for the modified CSP such that it is as similar to the previous one as possible, so that the number of people and resources affected can be kept to minimal.

To quote an example from [1], consider that there is a system that makes use of constraint satisfaction to schedule gates to arriving and departing flights in an airport. If the schedule is interrupted by unforeseen events such as bad weather, the gates assignment must be rearranged immediately and with as few modifications as possible in order to minimize the impact. Examples of other real-world problems include train rescheduling [7], operation scheduling for remote sensing satellites [39] and aircraft utilization problems [31].

1.4 Overview of the Thesis

The structure of the thesis is as follows. Chapter 2 reviews the major literature on the DCSP and algorithms for doing optimization on CSPs. DCSP algorithms for solution stability are distinguished from those for efficiency to provide a clearer comparison. As the stability problem is intrinsically an optimization problem, constrained optimization algorithms are also presented.

Chapter 3 gives the background knowledge for the thesis: the definition of the fuzzy CSP and how it can be solved using fuzzy GENET. A deficiency in fuzzy GENET, which we discovered while we were experimenting with it, is discussed. Due to this problem, a rectification is hence proposed.

Chapter 4 begins our contribution to the stability problem. We first illustrate how to model a stability problem as a fuzzy CSP and the rationale behind the

modelling, and then extend fuzzy GENET to fuzzy GENET(DCSP) so that it is able to find solutions that are more stable progressively. Experiments are performed on a wide range of randomly-generated binary CSPs, and the results are compared with optimal solutions.

Chapter 5 presents an enhancement to the scheme of the last chapter. The modelling process for stability problems with different distance requirements is unified by an n-ary fuzzy constraint called the *distance bound*. Fuzzy GENET(DCSP) is also enhanced further to fuzzy GENET(DCSP_2) to deal with the distance bound and for finding stable solutions more effectively. Fuzzy GENET(DCSP_2) is compared empirically with optimal solutions on small problems, and with fuzzy GENET(DCSP) on larger problems. Finally, we propose an idea to potentially increase the performance of fuzzy GENET(DCSP_2) based on the analysis of the results.

Chapter 6 concludes the thesis by summarizing our contributions and gives possible directions for future research.

Chapter 2

Related Work

The notion of dynamic constraint satisfaction problem can be traced back to Dechters [11], in which a CSP is represented as a constraint network, and a dynamic constraint network is defined as a sequence of static constraint networks each resulting from a change in the preceding one. While emphasizing on how to achieve belief maintenance of the constraint network in dynamically changing environments, Dechters have not presented a complete method to solve a dynamic CSP. In Sections 2.1 and 2.2, we survey existing DCSP algorithms, which are roughly divided into two main classes. The first class includes complete search algorithms that aim for efficiency, while the second class targets for solution stability. In Section 2.3, we survey algorithms for optimizing CSPs, since the stability problem can be viewed as an optimization problem. Lastly, we present an interesting real-life application called CASPER, which is created by the Jet Propulsion Laboratory.

2.1 Complete Search Algorithms

In this section, we focus on DCSP complete search algorithms, which are all based on backtrack tree-searching. As they become slower when the size of the search space grows, the major goal of these algorithms is to improve the

execution speed.

2.1.1 DnAC-4

Since the problem of proving the existence of a solution in a CSP is NP-complete, arc-consistency algorithms [25, 20, 24] are developed to prune away values that are irrelevant in any of the problem's solutions from variable domains. Arc-consistency methods are popular because they are effective in reducing the problem's search space. Bessière proposes DnAC-4 [2, 3], the first AC method to tackle the DCSP, which is based on AC-4 [24].

AC procedures, which only reduce the size of variable domains, are intrinsically incremental with respect to problem restrictions, because addition of constraints and variables will only lead to more inconsistent values being deleted. Therefore, an arc-consistent state can be achieved again by executing AC from the last consistent state. However, when dealing with problem relaxation, some of the removed values have to be *restored*, that is to be put back to the constraint network. It is because after some constraints or variables have been taken away from the problem, some domain values are no longer inconsistent. It is fine to restore more values than necessary, then add the extra constraints and variables and execute the AC procedure again. In other words, we can put back all the values that are originally present at the start of the problem. This is identical to solving the problem from scratch again. Although this approach works, obviously it is inefficient.

The idea of Bessière's DnAC-4 is to keep track of the *justification* of the values that are deleted during restrictions. A justification is the first constraint that makes a value without support and hence being deleted from its domain. With the help of the recorded justifications, domain values can be restored during relaxation.

However, DnAC-4 has an expensive worst-case space complexity and a bad average time complexity comparing with the subsequent methods.

2.1.2 AC|DC

Aiming at reducing the space complexity, Neveu *et al.* abandon the mechanism of reasons maintenance, and develop AC|DC [27]. The idea of AC|DC can be summarized in the following three steps, which are taken after a problem relaxation has occurred:

1. **Propose** : For the variables that connect to the deleted constraints, propose a set S of values that should be restored due to the removal of this constraint.

2. **Propagate** : Assume that all elements of the set S are restored, propagate the values throughout the whole constraint network to identify values in other variables that should also be restored due to the re-addition of values in S.

3. **Filter** : Add the restorable values identified in the last two steps, and then use any AC procedure to filter out any inconsistent values that have been added, since the set of values previously restored is over-estimated.

Compare with DnAC-4, AC|DC always perform poorer, although its execution time becomes faster and faster when the number of restored values increases. This is due to the inaccurate estimation of the restorable values which have to be removed again in the Filter step. At the sacrifice of efficiency, the space complexity of AC|DC can be reduced, because AC|DC does not need any data structures to store extra information like justifications.

2.1.3 DnAC-6

DnAC-6 [10] is an algorithm that strikes a balance between DnAC-4 and AC|DC in space complexity, and is less expensive than DnAC-4 in time complexity. DnAC-6 works like DnAC-4, in which it also keeps track of justifications of deleted values. However, it performs better than DnAC-4 because it benefits

by the feature of AC-6 [4], which considers a total ordering on values. If a value i in variable x is deleted, and is currently supporting value j of variable y, to see if j is still being supported by any other values, it is sufficient to check the values that are larger than i. In this case, i is called the *current support* of j. Due to this reason, the size of the set of lists to store justifications can be reduced, since it is no longer needed to record which values in variable x are supporting value j in variable y. It is enough to record the current support of j. This results in a smaller space complexity.

2.2 Algorithms for Stability

Research on stability problems starts from [1], and techniques have become more mature since then. More diversified techniques, such AC algorithms, stochastic local search and linear programming, have been employed to tackle the problem.

2.2.1 Bellicha

The approach of Bellicha *et al.* [1] is to transform the constraint graph of a CSP into a directed tree (if such transformation is possible) and take it as the input. And then it counts the minimal number of modifications needed for each pair (X, a), where a is the value that will be assigned to X. This is done by starting the procedure at the leaves of the tree, and then propagate their minimal numbers of modifications to the parent nodes. This method is only applicable to CSPs that can turn their constraint graphs into directed trees. If the constraint graph is cyclic, then such transformation is impossible. In this case, the graph is decomposed by the *cycle cutset method*, and then the algorithm is applied in each subgraph accordingly. However, the time complexity of the whole process for cyclic graphs is $O(md^{c+2})$, where m is the number of constraints, d the size of the largest domain, and c the size of the cutset. The drawback of this method is that it cannot be easily applied to any CSPs. There is also no empirical results

to verify its performance.

2.2.2 Dynamic Dynamic Backtracking[1]

Verfaillie and Schiex propose an extension of dynamic backtracking to solve DCSPs, and named the algorithm *dynamic dynamic backtracking* (*ddbt*) [38, 39]. Despite the name implies, dynamic backtracking [14] does not solve dynamic CSPs. The name of dynamic backtracking comes from the fact that the search can backtrack to a variable without unassigning other variables in between. However, dynamic backtracking has some good features that can be exploited on DCSPs. Dynamic backtracking records some sets that are called *eliminating explanations.* Eliminating explanations are defined as a pair (v, S), meaning that the variable that value v belongs to cannot take the value v because of the values already assigned to variables in set S. This mechanism helps dynamic backtracking not to erase the work done in the past.

Verfaillie and Schiex extended this notion and replaced the eliminating explanation with *variable eliminating explanation* and *constraint eliminating explanation.* Variable eliminating explanation is the same as eliminating explanation in the context of dynamic backtracking. Constraint eliminating explanation is similar to the variable counterpart, but it refers to the constraints that prohibit the assignment of a certain value rather than variables that have been previously assigned. The dynamic dynamic backtracking algorithm can be summarized in the following steps:

1. **Remove assignments** : Unassign variables that are suppressed by the newly added constraints, and create eliminating explanations at the same time.

2. **Remove variable eliminating explanations** : Remove the variable eliminating explanations that are related to the unassigned variables.

[1]This is not a typing mistake, there are two *Dynamic*s in the name.

3. **Remove constraint eliminating explanations** : Remove the constraint eliminating explanations that are related to the retracted constraints.

4. **Restart** : At this point a consistent starting point is created and the search can proceed again.

Dynamic dynamic backtracking has the notable feature that the previous solution and eliminating explanations are systematically reused in the new search, which may provide good results in terms of stability as well as efficiency. Randomly-generated CSPs of 16 variables, domain size of 13, constraint tightness ranging from 0.1 to 0.9 and constraint density from 0.2 to 0.9 are tested [39]. Experimental results confirm that by reusing the previous solutions, the distance between two successive solutions are about half of those attained by using conflict-directed backjumping and dynamic backtracking to solve the new CSP from scratch, but the solution quality is comparable to that of min-conflicts [22].

2.2.3 Wallace and Freuder

Wallace and Freuder have discussed stable solutions for DCSPs in [47], but their definition of stability is different from ours. They define a stable solution as one that is most likely to remain valid upon future CSP modifications, while our definition is to find a solution that is the most similar to the one in the previous CSP. Nevertheless, it is worth mentioning that their search scheme is based on min-conflicts hill-climbing [22], which is a kind of local search methods. In order to improve solution qualities, apart from the straightforward solution reuse, penalties are imposed on 'bad' values to direct the search away from them. Their initial experiments show that their strategies are generally effective in finding stable solutions, and in some cases can also balance the tradeoff between solution quality and efficiency quite well.

2.2.4 Unimodular Probing

Sakkout *et al.* develop an algorithm called *unimodular probing* [31, 30, 32] to reconfigure schedules in a changing environment with minimal perturbation to the existing schedules. The kind of scheduling problems they target for is the *kernel resource feasibility problem* (KRFP), which can be modelled as a CSP. The objective of KRFP is to find the start and end times of activities, such that temporal constraints (*e.g.* start time of activity $A \leq$ end time of activity $B + 10$ units of time) and resource constraints (quantities of available resource cannot be over-allocated) are satisfied. The famous job shop scheduling problem is a KRFP instance. In a changing environment, constraints can be added or deleted, it is optimal if a solution of minimal *perturbation* can be found, where perturbation is measured by a user defined function to calculate the difference between two complete assignments.

Unimodular probing is an algorithm that combines linear programming (LP) and constraint programming (CP). It works by using an LP solver on a subset of the constraints that possess the *total unimodular* (TU) property, and returns a solution, which is called a *unimodular probe*. If this probe violates any resource constraints, a violated constraint is selected among them and a new temporal constraint, which satisfies the TU property, is imposed. Since there may be several choices for a violated constraint, backtracking may be done at this point later. Addition of this new constraint prohibits some values to be assigned to the variables in this violated constraint. Local consistency methods are then applied to produce more TU constraints. The LP solver is then activated again. The procedure is repeated until the LP solver finds a probe that satisfies all the constraints; this probe is hence a solution. After a solution is found, branch and bound search can be conducted to find the optimal solution.

The algorithm has been tested on a large scale commercial application and the authors report it "significantly outperform rival algorithms" in [32], but due to the sensitivity of the data, they cannot publish the problems. Neverthe-

less, it performs better than CPLEX, a commercial mixed integer programming package, in their experiments on some benchmark problems.

2.2.5 Train Rescheduling

Due to unexpected events, a train timetable may become infeasible and has to be modified. Chiu *et al.* attempt to automate the process of rescheduling an infeasible train timetable [7] to generate a feasible one. They formulate the train rescheduling problem as a CSP, and derive an algorithm to perform the rescheduling based on a propagation-based constraint solver. Although efficiency of the algorithm is their main concern, as rescheduling must be accomplished in a timely manner, two definitions of optimality are also defined. These two optimality criteria are (1) *minimum-changes optimal*, which aims at a timetable with the least number of modified station visits, and (2) *minimum-delay optimal*, which wants to minimize the longest delay among all train visits. Variable and value ordering heuristics are proposed to try to fulfill the optimality criteria. A prototype system is done and real-life data are tested to confirm the feasibility of their proposed algorithms and heuristics.

2.3 Constrained Optimization Algorithms

The stability problem is a form of an optimization problem since the aim of stability problems is to minimize the distance between the new solution and the old one. In the following, we cover two stochastic local search algorithms for solving constrained optimization problems.

2.3.1 Guided Local Search

Guided local search (GLS) [41, 44, 42] is a general optimization technique for combinatorial optimization problems. Like its predecessor, GENET [8], and as its name implies, it looks for solutions by using local search. Optimization

is done by realizing *features* that are specific to the problem, and associates *costs* to each feature. For instance, in the travelling salesman problem (TSP), a feature could be "whether the candidate tour travels immediately from city A to city B," and the cost is the distance between cities A and B.

GLS has been tested on some well-known benchmark problems, such as TSP [44] and radio link frequency assignment problem (RLFAP) [43]. Experimental results show that it can find high quality and optimal solutions for TSP, and it has been able to discover better solutions than the best known ones for some RLFAP instances in 1998.

2.3.2 Anytime CSA with Iterative Deepening

Constrained Simulated Annealing (CSA) [46] is a global optimization algorithm that achieves asymptotic convergence to constrained global minima (CGM) with probability one for solving discrete constrained nonlinear programming problems (NLPs). In short, solving an NLP is to find the values for a vector of variables, such that some functions (constraints) are satisfied and the objective function is minimized. Based on CSA, Wah *et al.* develop CSA_{AT-ID} [45], the anytime CSA with iterative deepening. CSA_{AT-ID} consists of two main components. First, iterative deepening is performed by executing CSA with a set of geometrically increasing cooling schedules, each involving multiple runs of CSA. Second, the anytime property is realized by setting a new objective target f when a solution of quality f is found, so that better solutions can be found when more time is used to repeat the steps in the first component.

Wah *et al.* not only show that CSA_{AT-ID} is optimal, in the sense that the average time spent is up to an order of magnitude with respect to that of the original CSA with an optimal cooling schedule, but also show that CSA_{AT-ID} performs better than CSA as an anytime algorithm: CSA_{AT-ID} finds better solutions than CSA within a given time frame, and CSA_{AT-ID} spends one to two order less CPU time to find solutions of the same quality than CSA does.

2.4 A Real-life Application

In this section, we introduce a real-life application called CASPER, which aims at shortening the time for automatically replanning autonomous spacecrafts around updated information coming from execution monitoring.

As onboard computational resources in an autonomous spacecraft are typically limited, purposeful activities must be performed to ensure that long-term science and engineering goals are achieved. This requires planning in advance to avoid a series of shortsighted decisions. However, spacecraft plans may need to be modified due to fortuitous events such as early completion of observations and setbacks such as failure to acquire a guidestar for a science observation. It is advantageous if the planning process is more responsive to changes in the operations context since it would increase the overall time for which the spacecraft has a consistent plan. so that the spacecraft can keep busy working on the requested goals as long as a consistent plan exists.

To achieve a higher level of responsiveness in a dynamic planning situation, Chien *et al.* in the Jet Propulsion Laboratory utilize a continuous planning approach and implement a system called CASPER (Continuous Activity Scheduling Planning Execution and Replanning) [6, 5, 12]. Rather than considering planning a batch process in which a planner is presented with goals and an initial state, the planner has a current goal set, a plan, a current state, and a model of the expected future state. The planner is responsible for maintaining a consistent plan with the most current information. CASPER is built by using the ASPEN (Automated Scheduling and Planning ENvironment) framework [28], which employs iterative repair techniques [23] to enable incremental changes to the goals or the plan and then iteratively resolve any conflicts in the plan.

Chapter 3

Background

In this chapter, we introduce the definition of fuzzy constraint satisfaction problems (FCSPs), and review the local search algorithm fuzzy GENET for solving FCSPs. Fuzzy GENET forms the basis of our algorithm for solving stability problems. Moreover, during our experimentation, we discovered a deficiency in the original fuzzy GENET, which makes it unable to solve some FCSPs. A rectification to this problem is thus proposed.

3.1 Fuzzy Constraint Satisfaction Problems

The classical constraint satisfaction framework handles constraints as *crisp* entities, which means that they are either satisfied or unsatisfied. It is thus impossible to model situations involving partially (un)satisfied constraints. Fuzzy constraint satisfaction problem [29, 33, 21] extends classical CSP by modelling constraints as fuzzy relations in order to handle partial satisfaction of constraints.

Formally, a fuzzy constraint satisfaction problem is defined as follows:

Definition 3.1.1 (Fuzzy Constraint Satisfaction Problem) A fuzzy constraint satisfaction problem is a tuple (Z, D, C^f), where

- Z is a finite set of variables $\{z_1, z_2, \ldots, z_n\}$.
- D is a finite set of domains, each element D_i of which is the set of possible values for z_i.
- C^f is a finite (possibly empty) set of fuzzy constraints $\{c_1^f, \ldots, c_m^f\}$. Each fuzzy constraint $c_i^f \in C^f$ is a k-ary fuzzy relation R_i^f among variables $\{z_{i_1}, \ldots, z_{i_k}\}$, which is a subset of Z.

Definition 3.1.2 (Membership Function) The *membership function* $\mu_{R_i^f}$ of R_i^f defines the mapping:

$$\mu_{R_i^f} : D_{z_{i_1}} \times \cdots \times D_{z_{i_k}} \to [0, 1]$$

for assigning a value called the *degree of satisfaction* $\alpha_{\langle z_{i_1}, v_1\rangle \cdots \langle z_{i_k}, v_k\rangle} \in [0, 1]$ to each tuple $(v_1, \ldots, v_k) \in D_{z_{i_1}} \times \cdots \times D_{z_{i_k}}$.

The degree of satisfaction indicates the extent that the tuple $(v_1, \ldots, v_k)$ satisfies c_i^f. The larger the degree of satisfaction, the more $(v_1, \ldots, v_k)$ satisfies c_i^f. A value of 1 means $(v_1, \ldots, v_k)$ fully satisfies c_i^f and a value of 0 means it fully violates c_i^f. An intermediate value denotes a partial satisfaction.

In order to measure the overall satisfaction of an FCSP P and to compare how well the constraints are satisfied in aggregation when different values are assigned to the variables, a *global satisfaction degree* $\hat{\alpha}_{\langle z_1, v_1\rangle \cdots \langle z_n, v_n\rangle} \in [0, 1]$ is used. This global satisfaction degree is calculated by applying an *aggregate operator* f_α to the degrees of satisfaction of the tuples in P's constraints, one tuple for each constraint, as follows:

$$\hat{\alpha}_{\langle z_1, v_1\rangle \cdots \langle z_n, v_n\rangle} = f_\alpha(\mu_{R_1^f}(v_{1_1}, \ldots, v_{1_{n_1}}), \ldots, \mu_{R_m^f}(v_{m_1}, \ldots, v_{m_{n_m}})) \tag{3.1}$$

where n_j is the arity of R_j^f, v_{i_j} is the assigned value to variable z_{i_j}, and m is the number of fuzzy constraints in P.

Ruttkay [29] introduced 3 plausible aggregate operators:

1. minimum: $\min_{i=1}^{m} \mu_{R_i^f}(v_{i_1}, \ldots, v_{i_{n_i}})$,
2. product: $\prod_{i=1}^{m} \mu_{R_i^f}(v_{i_1}, \ldots, v_{i_{n_i}})$, and
3. average: $\frac{1}{m} \sum_{i=1}^{m} \mu_{R_i^f}(v_{i_1}, \ldots, v_{i_{n_i}})$.

The choice for an aggregate operator is problem-dependent.

Every FCSP is also associated with a user-specified *threshold* $\alpha_0 \in [0, 1]$ which defines the lowest acceptable global satisfaction degree of the problem. Hence a solution of an FCSP is a variable assignment $(v_1, v_2, \ldots, v_n)$ which fulfills the requirement $\hat{\alpha}_{\langle z_1, v_1 \rangle \cdots \langle z_n, v_n \rangle} \geq \alpha_0$.

3.2 Fuzzy GENET

Fuzzy GENET [49, 50] is a stochastic local search algorithm for solving *binary* FCSPs, that is one with only unary and binary constraints. It is achieved by incorporating the concept of fuzziness into the CSP solver GENET [8]. Fuzzy GENET consists of two parts: one is the network architecture and the other is the convergence procedure. The network architecture concerns the representation of an FCSP, while the convergence procedure includes an iterative improvement algorithm that is responsible for finding solutions.

3.2.1 Network Architecture

A fuzzy GENET network consists of 3 main components: clusters, label nodes and connections. Given an FCSP P, each variable in P is represented in the network as a *cluster* which consists of one or more *label nodes* that correspond to the values in the domain of this variable. A label node for value j in the domain of variable i is denoted as $\langle i, j \rangle$. Each label node has two states: *on* and *off*. A label node in the on state is described as an on-node; likewise, one in the off state is an off-node. An on-node indicates that its corresponding value is assigned to the variable associated with the cluster that the on-node belongs

to. Obviously there can only be exactly one and only one node that is being turned on in a cluster at any time and all the other nodes must be off, because you cannot assign more than one value to a variable at the same time.

For each binary fuzzy constraint between variables i and k, a *weighted connection* is placed between each pair of label nodes of the two clusters of variables i and k. In the case of a unary fuzzy constraint, the connections are linking to the label nodes themselves. There are two values associated with each weighted connection. One is the degree of satisfaction, which has been described in Definition 3.1.2, and the other is the *weight*. The degree of satisfaction is fixed at the time when the network is built and will never change, but the weight may be decremented during the *learning phase* in the convergence procedure. For the connection between label nodes $\langle i,j\rangle$ and $\langle k,l\rangle$, its degree of satisfaction is $\alpha_{\langle i,j\rangle\langle k,l\rangle}$ and its weight, denoted as $W_{\langle i,j\rangle\langle k,l\rangle}$, is initially set to $\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1$. For clarity, connections with $W_{\langle i,j\rangle\langle k,l\rangle} = 0$ (*i.e.* $\alpha_{\langle i,j\rangle\langle k,l\rangle} = 1$) are usually not shown in the network graph explicitly because no connection implies totally no conflicts between the two label nodes.

The *output* $O_{\langle i,j\rangle}$ of the node $\langle i,j\rangle$ is 1 if $\langle i,j\rangle$ is on and 0 otherwise. The node $\langle i,j\rangle$ also has an *input* $I_{\langle i,j\rangle}$, which is calculated as the weighted sum of the outputs of all label nodes connecting to $\langle i,j\rangle$:

$$I_{\langle i,j\rangle} = \sum_{\langle k,l\rangle \in \mathcal{A}(\langle i,j\rangle)} W_{\langle i,j\rangle\langle k,l\rangle} O_{\langle k,l\rangle} \tag{3.2}$$

where $\mathcal{A}(\langle i,l\rangle)$ is the set of all label nodes that are connected to $\langle i,j\rangle$.

We give an example to illustrate all the concepts presented so far. The problem we use is a slightly modified version of the Robot Dressing Problem [13, 49]. In this problem, a robot tries to select matching clothes while getting dressed in the morning. The robot has only a minimal wardrobe: sneakers or Cordovans for footwear, a white and a dark green shirt, and three pairs of trousers: blue slacks, grey slacks, and denims. The robot has been told that: the sneakers only go with the denims; the Cordovans only go with the gray slacks

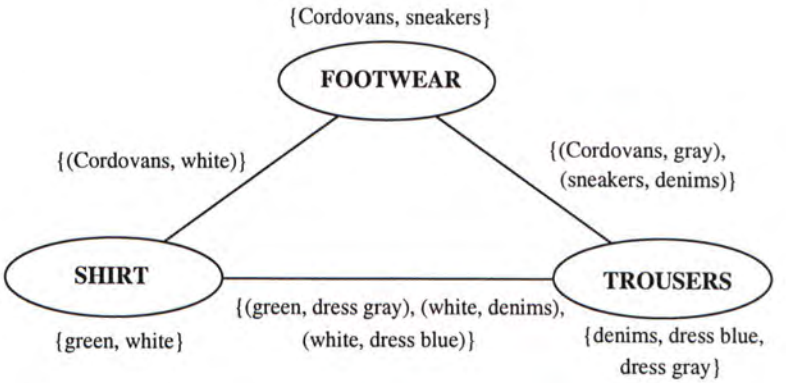


Figure 3.1: Constraint graph of the Robot Dressing Problem

and the white shirt; the white shirt goes with either denims or blue trousers; and the green shirt only goes with the gray trousers. The constraint graph for this CSP is depicted in Figure 3.1. Since the problem is over-constrained, it has no solution. The problem is then relaxed and modelled as an FCSP with the degrees of satisfaction (and the corresponding initial weights) specified in Table 3.1. The first column *compound label* is a shorthand notation $\langle i, j\rangle\langle k, l\rangle$ to denote a tuple (j, l) of the constraint with variables i and k. Combining all the information, we get a fuzzy GENET network in Figure 3.2.

3.2.2 Convergence Procedure

The convergence procedure of fuzzy GENET is shown in Algorithm 3.1. First, a label node in each cluster is selected to be turned on, and the weights of all the tuples are initialized according to their predefined degrees of satisfaction as described in Section 3.2.1. The algorithm will then enter the convergence cycle, which will only be exited when the global satisfaction degree of the FCSP is larger than or equal to the threshold α_0, in which case a solution is found. In each cycle, states of all label nodes are updated *asynchronously in parallel* for each cluster. In a sequential implementation of fuzzy GENET, asynchronous update is done by updating each cluster in a predefined order. The update

compound label t	α_t	W_t
⟨Shirt, green⟩ ⟨Trousers, denims⟩	0.9	−0.1
⟨Shirt, green⟩ ⟨Trousers, blue⟩	0.7	−0.3
⟨Shirt, green⟩ ⟨Trousers, gray⟩	1.0	0.0
⟨Shirt, white⟩ ⟨Trousers, denims⟩	1.0	0.0
⟨Shirt, white⟩ ⟨Trousers, blue⟩	1.0	0.0
⟨Shirt, white⟩ ⟨Trousers, gray⟩	0.5	−0.5
⟨Footwear, Cordovans⟩ ⟨Trousers, denims⟩	0.1	−0.9
⟨Footwear, Cordovans⟩ ⟨Trousers, blue⟩	0.8	−0.2
⟨Footwear, Cordovans⟩ ⟨Trousers, gray⟩	1.0	0.0
⟨Footwear, sneakers⟩ ⟨Trousers, denims⟩	1.0	0.0
⟨Footwear, sneakers⟩ ⟨Trousers, blue⟩	0.1	−0.9
⟨Footwear, sneakers⟩ ⟨Trousers, gray⟩	0.7	−0.3
⟨Shirt, green⟩ ⟨Footwear, Cordovans⟩	0.6	−0.4
⟨Shirt, green⟩ ⟨Footwear, sneakers⟩	0.1	−0.9
⟨Shirt, white⟩ ⟨Footwear, Cordovans⟩	1.0	0.0
⟨Shirt, white⟩ ⟨Footwear, sneakers⟩	0.2	−0.8

Table 3.1: Degrees of satisfaction and weights of the relaxed robot dressing problem

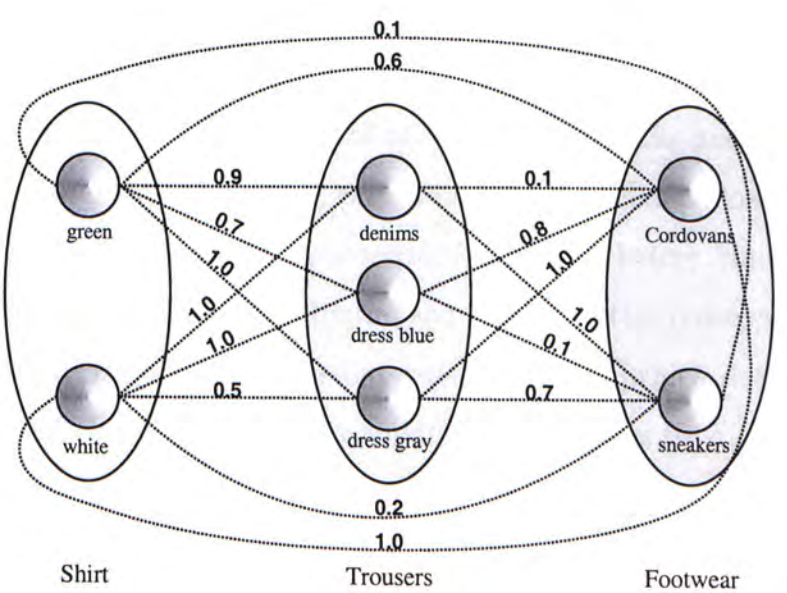


Figure 3.2: The Fuzzy GENET network of the relaxed robot dressing problem

1: randomly turn on a label node per cluster
2: **for** each $W_{\langle i,j\rangle\langle k,l\rangle}$ **do**
3: $\quad W_{\langle i,j\rangle\langle k,l\rangle} \leftarrow \alpha_{\langle i,j\rangle\langle k,l\rangle} - 1$
4: **end for**
5: **while** $\alpha_{P,\langle z_1,v_1\rangle\cdots\langle z_n,v_n\rangle} < \alpha_0$ **do**
6: $\quad$ **for** each cluster **do**
7: $\quad\quad$ *(asynchronously in parallel)*
8: $\quad\quad$ calculate the input of each label node
9: $\quad\quad$ turn on the label node with maximum input
10: $\quad$ **end for**
11: $\quad$ **if** local maximum reached **then**
12: $\quad\quad$ *(Heuristic learning rule)*
13: $\quad\quad$ **for** each $W_{\langle i,j\rangle\langle k,l\rangle}$ **do**
14: $\quad\quad\quad$ update $W_{\langle i,j\rangle\langle k,l\rangle}$: $W^{new}_{\langle i,j\rangle\langle k,l\rangle} = W^{old}_{\langle i,j\rangle\langle k,l\rangle} + O^{old}_{\langle i,j\rangle} O^{old}_{\langle k,l\rangle}(\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1)$
15: $\quad\quad$ **end for**
16: $\quad$ **end if**
17: **end while**

Algorithm 3.1: The fuzzy GENET convergence procedure

always turns on the label node that takes the maximum input in each cluster, and all other label nodes are turned off. If there are more than one node that receive the same maximum input in a cluster, then one of the nodes is chosen randomly, unless one of them is already on in the last cycle. In the latter case, the node remains on in the next cycle.

If, in a certain cycle, the states of all the nodes remain unchanged, fuzzy GENET is in a state known as *local maximum*. Solutions can no longer be found merely by using the state updating procedure illustrated before, since the inputs of all on-nodes are already the maximum and thus no better move can be found. In this situation, the *heuristic learning rule* is applied, which decrements the weights of those connections that are linking two on-nodes by $\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1$:

$$W^{new}_{\langle i,j\rangle\langle k,l\rangle} = W^{old}_{\langle i,j\rangle\langle k,l\rangle} + O^{old}_{\langle i,j\rangle} O^{old}_{\langle k,l\rangle}(\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1) \tag{3.3}$$

The current state will no longer be a local maximum since there will be other nodes that have higher inputs, possibly after applying the heuristic learning rule more than once consecutively.

Connected nodes should not be on simultaneously as far as possible. The heuristic learning rule tries to penalize such occurrences when the network is trapped in a local maximum. This makes the penalized pairs of on-nodes $\langle i, j \rangle$ and $\langle k, l \rangle$ less likely be turned on simultaneously again in the future. This effect favours the selection of nodes with higher inputs, which are more possibly be in the solutions.

3.3 Deficiency in Fuzzy GENET

While we were carrying out the experiments with fuzzy GENET, we discovered its inability to solve a certain kind of FCSPs, having the variables oscillate from state to state indefinitely.

The deficiency described is manifested by solving the following FCSP:

- The FCSP consists of four variables z_1, z_2, z_3 and z_4, and they all have the domain $\{0, 1, 2, 3\}$.
- There are two crisp constraints between any two variables: "$z_i \neq z_j$" and "$|z_i - z_j| \neq j - i$".
- 4 more unary fuzzy constraints "$z_i = 0$" for $i \in \{1, 2, 3, 4\}$, one for each variable. Degrees of satisfaction are defined as

Value	Degree of satisfaction
0	1.00
1	0.10
2	0.10
3	0.10

- The threshold is 0.10.
- Initial variable assignment is $(0, 1, 0, 1)$.
- During the convergence procedure, always select the node with the smallest value when there are more than one node that have the maximum

input, unless the input of the current on-node is also the maximum: in this case, the on-node is remain to be on. (This is a special case of the usual "break-tie-randomly" strategy used in fuzzy GENET.)

The first 10 iterations of the execution are summarized in Figure 3.3. The underlined numbers correspond to the on-nodes of the variables at the *end* of that iteration, whereas the inputs of nodes shown are *not* the inputs at the end of that iteration, but are those during the node selection step of that particular variable, that is, during steps 5 to 8 in Algorithm 3.1. For instance, in iteration 1, we show $-2.0, -2.9, -2.9, -0.9$ as inputs of nodes 0 to 3 in variable z_0, which are what we get while we are choosing the best node, but at the end of iteration 1, their inputs become $-2.0, -2.9, -1.9, -0.9$ due to changes of on-nodes in the other variables.

The inability of fuzzy GENET can be attributed to the fact that some constraints are penalized unnecessarily, such that the cost surface is distorted to the extent that some local maxima originally corresponding to solutions are destroyed. As these solutions are no longer located at local maxima, they cannot be found by the convergence procedure. This can be seen in Figure 3.3(i). According to the problem specification, the assignment $(2, 0, 3, 1)$ is obviously a solution. However, in the next iteration (Figure 3.3(j)) node 0 of variable z_0 is turned on in favour of node 2 because the input of node 0 is the highest among all the others. This shows that the search moved away from the solution state and did not settle there. As a result, solutions were not found.

Actually this also happens toward the end of iteration 4 (Figure 3.3(d)), though less obviously: node 0 in variable z_3 is chosen instead of staying at the current node then, that is node 1, due to the higher input of node 0. If the state of variable z_3 remains unchanged, a solution has already been found, which is $(2, 0, 3, 1)$.

Detailed examination of past iterations sheds light on the cause of this cost surface distortion. Consider the fact that selected nodes must have the highest

inputs, node 1 of variable z_3 in iteration 4 and node 2 of variable z_0 in iteration 9 must have been penalized inappropriately in the past such that they are less appealing. Their inputs at the moment when they move away from the "solution states" are both -1.8. By tracing the past iterations, it reveals that these penalties are accumulated by penalizing the unary constraint "$z_3 = 0$" during the learning phase in iteration 2 (Figure 3.3(b)) when $z_3 = 1$ and the unary constraint "$z_0 = 0$" during the learning phase in iteration 5 (Figure 3.3(e)) when $z_0 = 2$. If they are not penalized, their inputs will both be -0.9 and then will remain in their original states in these two iterations, and hence solution would be found.

This indicates that penalizing them at those moments is problematic.

The purpose of penalization in GENET is to alter the cost surface in order to distinguish between the tuples that are promising and those that should be avoided. As such it is not helpful, or even harmful (as shown in the example), to penalize the constraints that are not really violated. Refer to the example again. In iteration 2, the constraint "$z_3 = 0$" should not be regarded as violated because $z_3 = 1$ can rightfully be part of the solution, as long as all the hard constraints are satisfied and the global satisfaction degree is above the threshold. The same applies to the case in iteration 5. However, fuzzy GENET just tries to penalize all connecting connections whose degrees of satisfaction is not equal to 1, which ignores the effects brought by the aggregate function and the threshold of FCSPs. In Section 3.4, we will generalize this idea and propose a remedy to this behaviour.

3.4 Rectification of Fuzzy GENET

To correct the problematic behaviour of fuzzy GENET described in the last section, we propose to change the conditions under which the constraints are penalized. Before doing that, we need to first introduce the concept of *contribu-*

Variable	Input of node			
	0	1	2	3
z_0	-2.0	-2.9	-2.9	-0.9
z_1	-1.0	-2.9	-1.9	-2.9
z_2	-2.0	-3.9	-1.9	-1.9
z_3	-2.0	-1.9	-2.9	-2.9

(a) Iteration 1

Variable	Input of node			
	0	1	2	3
z_0	-2.0	-2.9	-1.9	-0.9
z_1	-1.0	-2.9	-2.9	-2.9
z_2	-2.0	-3.9	-1.9	-1.9
z_3	-2.0	-1.9	-2.9	-2.9

(b) Iteration 2 (local maximum)

Variable	Input of node			
	0	1	2	3
z_0	-2.0	-2.9	-1.9	-1.8
z_1	-1.0	-2.9	-2.9	-2.9
z_2	-2.0	-3.9	-3.8	-1.9
z_3	-2.0	-1.8	-2.9	-2.9

(c) Iteration 3

Variable	Input of node			
	0	1	2	3
z_0	-1.0	-3.9	-0.9	-2.8
z_1	-1.0	-2.9	-2.9	-2.9
z_2	-3.0	-2.9	-4.8	-0.9
z_3	-1.0	-1.8	-3.9	-1.9

(d) Iteration 4

Variable	Input of node			
	0	1	2	3
z_0	-2.0	-2.9	-0.9	-3.8
z_1	-1.0	-1.9	-3.9	-2.9
z_2	-3.0	-2.9	-2.8	-0.9
z_3	-1.0	-1.8	-3.9	-1.9

(e) Iteration 5 (local maximum)

Variable	Input of node			
	0	1	2	3
z_0	-2.0	-2.9	-1.8	-3.8
z_1	-2.0	-1.9	-3.9	-2.9
z_2	-3.0	-2.9	-3.8	-1.8
z_3	-1.0	-1.8	-2.9	-2.9

(f) Iteration 6

Variable	Input of node			
	0	1	2	3
z_0	-2.0	-2.9	-2.8	-3.8
z_1	-3.0	-1.9	-2.9	-1.9
z_2	-3.0	-2.9	-3.8	-1.8
z_3	-1.0	-2.8	-1.9	-3.9

(g) Iteration 7

Variable	Input of node			
	0	1	2	3
z_0	-2.0	-2.9	-2.8	-3.8
z_1	-3.0	-1.9	-2.9	-1.9
z_2	-3.0	-2.9	-3.8	-1.8
z_3	-1.0	-2.8	-1.9	-3.9

(h) Iteration 8 (local maximum)

Variable	Input of node			
	0	1	2	3
z_0	-4.0	-2.9	-2.8	-3.8
z_1	-2.0	-2.8	-3.9	-2.9
z_2	-3.0	-2.9	-2.8	-2.7
z_3	-2.0	-1.8	-3.9	-1.9

(i) Iteration 9

Variable	Input of node			
	0	1	2	3
z_0	-1.0	-3.9	-1.8	-2.8
z_1	-1.0	-4.8	-1.9	-2.9
z_2	-3.0	-2.9	-4.8	-2.7
z_3	-4.0	-1.8	-2.9	-2.9

(j) Iteration 10

Figure 3.3: An example showing the inadequacy of fuzzy GENET

tion of a constraint to the violation of an FCSP. This concept identifies whether or not a constraint is responsible for making the FCSP unsatisfiable. Having this knowledge, only the constraints that actually contribute to the FCSP's unsatisfiability are penalized in the learning phase.

Definition 3.4.1 Given a constraint $c_i^f(z_{i_1}, z_{i_2}, \ldots, z_{i_m})$, where z_{i_j} is assigned by v_{i_j}. c_i^f contributes to the violation of the FCSP iff

$$\nexists \text{ a tuple } (v_1, v_2, \ldots, v_n) \in D'_{z_1} \times D'_{z_2} \times \cdots \times D'_{z_n}$$

$$\text{where } D'_{z_k} = \begin{cases} \{v_k\} & \text{if } z_k \in \{z_{i_1}, z_{i_2}, \ldots, z_{i_m}\}, \\ D_{z_k} & \text{otherwise.} \end{cases} \tag{3.4}$$

such that $\hat{\alpha}_{\langle z_1, v_1\rangle \cdots \langle z_n, v_n\rangle} \geq \alpha_0$.

Consider the case in Figure 3.3(b) again. If contribution is considered, the constraint "$z_3 = 0$" will not be penalized because there exists a tuple $(2, 0, 3, 1)$ such that the global satisfaction degree of the FCSP is 0.10, that is $\hat{\alpha}_{\langle z_1, v_1\rangle \cdots \langle z_n, v_n\rangle} \geq \alpha_0$. We can conclude that the constraint "$z_3 = 0$" is not contributing to the violation of the FCSP.

The modified fuzzy GENET that makes use of Definition 3.4.1 is outlined in Algorithm 3.2. Specifically, line 13 of the algorithm is altered to overcome the mentioned deficiency in the original fuzzy GENET.

However, most of the time it is practically infeasible to check all possible tuples because the number of tuples is large. We should exploit the nature of

1: randomly turn on a label node per cluster
2: **for** each $W_{\langle i,j\rangle\langle k,l\rangle}$ **do**
3: $\quad W_{\langle i,j\rangle\langle k,l\rangle} \leftarrow \alpha_{\langle i,j\rangle\langle k,l\rangle} - 1$
4: **end for**
5: **while** $\hat{\alpha}_{\langle z_1,v_1\rangle\cdots\langle z_n,v_n\rangle} < \alpha_0$ **do**
6: $\quad$ **for** each cluster **do**
7: $\quad\quad$ *(asynchronously in parallel)*
8: $\quad\quad$ calculate the input of each label node
9: $\quad\quad$ turn on the label node with maximum input
10: $\quad$ **end for**
11: $\quad$ **if** local minimum reached **then**
12: $\quad\quad$ *(Heuristic learning rule)*
13: $\quad\quad$ **for** each $W_{\langle i,j\rangle\langle k,l\rangle}$ that contributes to the FCSP violation **do**
14: $\quad\quad\quad$ update $W_{\langle i,j\rangle\langle k,l\rangle}$: $W^{new}_{\langle i,j\rangle\langle k,l\rangle} = W^{old}_{\langle i,j\rangle\langle k,l\rangle} + O^{old}_{\langle i,j\rangle} O^{old}_{\langle k,l\rangle}(\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1)$
15: $\quad\quad$ **end for**
16: $\quad$ **end if**
17: **end while**

Algorithm 3.2: The rectified fuzzy GENET convergence procedure

the aggregate operator that is being used in the FCSP to reduce the number of tuples to check. For instance, if minimum is being used, it is sufficient to check whether the degree of satisfaction of the tuple of the constraint c_i^f is smaller than α_0. If it is, then we can conclude that c_i^f is contributing to the violation of the FCSP, because no matter what the degrees of satisfaction of the other tuples are, $\hat{\alpha}_{\langle z_1,v_1\rangle\cdots\langle z_n,v_n\rangle}$ is always smaller than α_0.

Chapter 4

Using Fuzzy GENET for Solving Stability Problems

In this chapter, we propose a scheme to model a stability problem as an FCSP and solve it by a specialized fuzzy GENET that we call fuzzy GENET(DCSP). We choose fuzzy GENET as the basis of our solver since it is substantially faster than complete search methods. Although optimal solutions are not guaranteed, due to the "hill-climbing" nature of the convergence procedure in fuzzy GENET, solutions of high stability can still be achieved.

4.1 Modelling Stability Problems as FCSPs

In view of the fact that the degree of satisfaction assigned to each tuple $(v_{z_1}, \ldots, v_{z_k})$ of a fuzzy constraint c^f indicates the extent that this tuple satisfies c^f, these degrees of satisfaction can be used to represent the user's preference on the tuples. By making use of this property, we can create some artificial fuzzy constraints to indicate which tuples and values are more beneficial to minimizing the distance, thus enhancing the stability of the solution of a restricted CSP.

Hence, we attempt to impose supplementary fuzzy constraints into a CSP after it is modified, and assign degrees of satisfaction to the tuples of these new

fuzzy constraints such that they reflect how stable they will give rise to the solution. Higher degrees of satisfaction are assigned to the tuples which involve in solutions of better quality, and lower degrees of satisfaction are assigned to those that involve in poorer solutions. Moreover, the fuzzy constraints should be modelled in such a way that the global satisfaction degree will be larger then or equal to the threshold, so that there will not be any subversive effect to the solution space of the original problem.

Recall that a DCSP is a sequence of CSPs $\{P_0, \ldots, P_i, P_{i+1}, \ldots\}$. To achieve a stable solution, $\boldsymbol{s}^{i+1}$, for P_{i+1} with respect to the solution $\boldsymbol{s}^i$ of P_i, we add supplementary fuzzy constraints C^f into P_{i+1} to form a new FCSP P^f_{i+1}. The role of the supplementary fuzzy constraints is to encourage the variables to select the values so as to minimize the distance between $\boldsymbol{s}^i$ and $\boldsymbol{s}^{i+1}$. The supplementary constraints must be fuzzy because they *can* optionally be violated. Otherwise some solutions may be lost.

Since the meaning of stability varies from problems to problems, the distance metrics of the problems are defined differently. As a result, there does not exist a universal way to model the supplementary fuzzy constraints, the membership function and the threshold of the resultant FCSP. However, the guideline is to define them in order to differentiate the tuples of the supplementary constraints according to the quality of the solutions that they belong to. This is further explained in the following two examples, which show how this guideline is applied to complete the modelling.

Example 4.1.1 Suppose we want to measure the number of different variable assignments between $\boldsymbol{a} = (a_1, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, \ldots, b_n)$. We use the *Hamming distance* d_H as the distance function:

$$d_H(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i \in \{0,\ldots,n\}} \delta(a_i, b_i), \qquad \text{where} \quad \delta(a_i, b_i) = \begin{cases} 1 & \text{if } a_i \neq b_i, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

After CSP P_i is restricted to become P_{i+1} due to the introduction of new

constraints, supplementary fuzzy constraints of the following form can be added into P_{i+1} to try to enforce the stability of the new solution of P_{i+1}:

$$c^f_{new_k} : z_k = s^i_k, \tag{4.2}$$

where s^i_k is the value of z_k in the solution of the preceding problem P_i.

The unary fuzzy constraint $c^f_{new_k}$ is actually a fuzzy relation $R^f_{new_k}$ with a single attribute acting on variable z_k, the membership function of $R^f_{new_k}$ can be defined as follows:

$$\mu_{R^f_{new_k}}(a) = \begin{cases} 1 & \text{if } a = s^i_k, \\ \alpha & \text{otherwise.} \end{cases} \tag{4.3}$$

where $\alpha \in [0, 1)$.

For instance, if the assignment of z_1 in the solution of P_1 is 2, and the domain of z_1 is $\{1, 2, 3, 4, 5\}$. The list of degrees of satisfaction produced by its membership function is as follows ($\alpha = 0.50$ in this case):

Tuple	Degree of satisfaction
(1)	0.50
(2)	1.00
(3)	0.50
(4)	0.50
(5)	0.50

Table 4.1: Degrees of satisfaction of Example 4.1.1

The fuzzy constraints in (4.2) suggest that it is the best for the variables to take the values that have been assigned to them in the preceding solution. Because the tuple (s^i_k) has the highest degree of membership 1.0, it is more preferable than all the other tuples. This is a straightforward way to minimize the distance. However, constraints introduced into P_i may prohibit variables to take their values from the preceding solution. Thus, the degrees of satisfaction of the supplementary fuzzy constraints and the threshold of the FCSP must be set so as to allow the violation of these fuzzy constraints.

For the FCSP in Example 4.1.1, the minimum function should be used as the aggregate operator and the threshold α_0 be set to α, because α is the highest acceptable global satisfaction degree to make it possible to violate the supplementary fuzzy constraints[1], while preventing it from violating the original hard constraints. This setting of parameters ensures that all original constraints shall be fully satisfied and the supplementary fuzzy constraints can just be partially satisfied.

Example 4.1.2 Suppose we are dealing with a CSP with only integer domains. In this example we use *Manhattan distance* d_M between the variable assignments $\boldsymbol{a} = (a_1, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, \ldots, b_n)$ as:

$$d_M(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} |a_i - b_i| \tag{4.4}$$

This distance metric is suitable for stability problems that want to choose values which are as *close* to those in the preceding solution as possible, in the sense that the smaller the absolute difference between two values, the better.

After new constraints are introduced into CSP P_i to become P_{i+1}, the same supplementary fuzzy constraints as shown in (4.2) of Example 4.1.1 are added, but the membership function of the constraint is defined differently, as follows:

$$\mu_{R^f_{new_k}}(a) = 1 - \frac{|a - s^i_k|}{\max(D_k) - \min(D_k) + 1} \tag{4.5}$$

where s^i_k is the value of z_k in $\boldsymbol{s}^i$, the solution of P_i.

Using the scenario in Example 4.1.1 again, the degrees of satisfaction of z_1 are listed in Table 4.2. Alternatively, it is shown graphically in Figure 4.1. As it can be seen from the figure, the choice of (4.5) fits nicely to the Manhattan distance: when the value goes farther away from the assignment in the preceding solution (that is the distance becomes greater), the corresponding degree of satisfaction decreases.

[1]Actually α_0 can be any value in $(0, \alpha]$, because no global satisfaction degree $\hat{\alpha}_{\langle z_1, v_1\rangle \cdots \langle z_n, v_n\rangle}$is in $(0, \alpha)$ and hence if $\alpha_0 \in (0, \alpha]$, $\hat{\alpha}_{\langle z_1, v_1\rangle \cdots \langle z_n, v_n\rangle} \geq \alpha \Leftrightarrow \hat{\alpha}_{\langle z_1, v_1\rangle \cdots \langle z_n, v_n\rangle} \geq \alpha_0$, that is, the solution condition is unchanged.

Tuple	Degree of satisfaction
(1)	0.80
(2)	1.00
(3)	0.80
(4)	0.60
(5)	0.40

Table 4.2: Degrees of satisfaction of Example 4.1.2

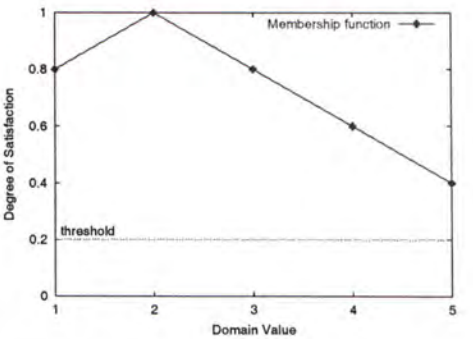


Figure 4.1: Manhattan distance membership function of Example 4.1.2

The aggregate operator should be the minimum function and the threshold should be the lowest value of $\mu_{R^f_{new_k}}(z_k)$ among all z_k. The reason for choosing them is the same as that in the last example. The threshold cannot be set higher than 0.4 in the example, otherwise some of the domain values will be excluded from selection, which may result in loss of solutions.

Example 4.1.3 To present all the concepts that we have discussed so far, here is a complete running example.

Suppose we have a stability problem $(\boldsymbol{s}^i, P_i, P_{i+1}, d)$, where P_i has three variables x, y, and z, with domains $\{1, 2, 3, 4, 5\}$, and consists of the following two constraints (equations):

$$x + y + z = 10 \tag{4.6}$$
$$x + y - z = 6 \tag{4.7}$$

There are many possible solutions for P_i. Suppose we arrive at the solution $\langle x, 4\rangle\langle y, 4\rangle\langle z, 2\rangle$, which is denoted as $\boldsymbol{s}^i$, by using any technique (whichever is not important in illustrating this example).

After $\boldsymbol{s}^i$ has been found, the following new constraint is added into P_i:

$$x + z = 7 \tag{4.8}$$

As a result, the new CSP, denoted as P_{i+1}, has totally three constraints:

(x)	1	2	3	4	5
$\mu_{R^f_{x=4}}(x)$	0.10	0.40	0.70	1.00	0.70
(y)	1	2	3	4	5
$\mu_{R^f_{y=4}}(y)$	0.10	0.40	0.70	1.00	0.70
(z)	1	2	3	4	5
$\mu_{R^f_{z=2}}(z)$	0.70	1.00	0.70	0.40	0.10

Table 4.3: Degrees of satisfaction of $x = 4$, $y = 4$, and $z = 2$

(4.6), (4.7), and (4.8). Note that the variables and their domains have not been altered.

According to our modelling scheme, we do not solve P_{i+1} directly to achieve stable solutions. Instead, we introduce some supplementary fuzzy constraints into P_{i+1} to produce P^f_{i+1}, and then solve P^f_{i+1} by fuzzy GENET. Suppose the d in $(\boldsymbol{s}^i, P_i, P_{i+1}, d)$ is the Manhattan distance, these supplementary fuzzy constraints will be introduced into P_{i+1}:

$$x = 4 \tag{4.9}$$
$$y = 4 \tag{4.10}$$
$$z = 2 \tag{4.11}$$

The degrees of satisfaction of (4.9) to (4.11) are defined as in Table 4.3 according to (4.5) when $\alpha_0 = 0.1$.

Thus, P^f_{i+1} have three more new constraints than P_{i+1}, and all of them are fuzzy. Variables and their domains have not been altered. Now, everything is prepared. We then use fuzzy GENET, with the initial values of the variables the same as in $\boldsymbol{s}^i$, to solve P^f_{i+1}.

4.2 Extending Fuzzy GENET for Solving Stability Problems

FCSPs created by the method in Section 4.1 can be solved by any FCSP solver. However, the modelling strategy is designed specifically for the fuzzy GENET FCSP solver to obtain more stable solutions. By making some modifications to fuzzy GENET and use this specialized version to solve the FCSPs, solutions of even higher quality can be obtained. The enhanced algorithm is shown in Algorithm 4.1, which we call it

initialize states of nodes as the last solution
for each $W_{\langle i,j\rangle\langle k,l\rangle}$ **do**
  $W_{\langle i,j\rangle\langle k,l\rangle} \leftarrow \alpha_{\langle i,j\rangle\langle k,l\rangle} - 1$
end for
$d \leftarrow$ max distance + 1
while stopping criteria not met and $d \neq 0$ **do**
  while $\alpha_P < \alpha_0$ **do**
    for each cluster **do**
      (asynchronously in parallel)
      calculate the input of each label node
      turn on the node with maximum input
    end for
    if local maximum reached **then**
      for each $W_{\langle i,j\rangle\langle k,l\rangle}$ that contributes to the FCSP violation **do**
        update $W_{\langle i,j\rangle\langle k,l\rangle}$: $W^{new}_{\langle i,j\rangle\langle k,l\rangle} = W^{old}_{\langle i,j\rangle\langle k,l\rangle} + O^{old}_{\langle i,j\rangle} O^{old}_{\langle k,l\rangle} (\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1)$
      end for
    end if
  end while
  if current distance $< d$ **then**
    record solution
    $d \leftarrow$ current distance
  end if
  force update $W_{\langle i,j\rangle\langle k,l\rangle}$: $W^{new}_{\langle i,j\rangle\langle k,l\rangle} = W^{old}_{\langle i,j\rangle\langle k,l\rangle} + O^{old}_{\langle i,j\rangle} O^{old}_{\langle k,l\rangle} (\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1)$
end while

Algorithm 4.1: The fuzzy GENET convergence procedure for DCSP — fuzzy GENET(DCSP)

The main difference between the original fuzzy GENET and fuzzy GENET(DCSP) is that, for the original fuzzy GENET, the procedure stops

once a solution is found, but fuzzy GENET(DCSP) does not. Instead, it will continue to look for a solution that is more stable than the solutions that have been found so far. Since the optimum of a problem is not known beforehand, when a better solution is found, fuzzy GENET(DCSP) moves on to look for an even better solution, if possible. This strategy improves fuzzy GENET in terms of solution quality, since solutions found by fuzzy GENET(DCSP) must be at least of the same quality as those found by fuzzy GENET.

However, simply restarting the search after a solution is found is futile because the solutions found are located at local maxima. As there is no better neighbourhood to move to at a local maximum, it will continue to be stuck there no matter how many times the search is restarted. The strategy employed by fuzzy GENET(DCSP) is to forcibly *penalize* the supplementary fuzzy constraints right after a solution is found and before restarting the search (Algorithm 4.1, line 23). The purpose of this penalization step is to disturb the cost surface a little in order to bring the search out of the local maximum, so that there are chances for it to migrate to its neighbourhood and proceed to look for better solutions.

To see how this works, let us consider a CSP P of 4 variables, z_0, z_1, z_2, z_3, with a known solution $(0, 0, 0, 0)$. Later, new constraints are introduced into P to make it P'. By modelling it with the settings in Example 4.1.1 with $\alpha = 0.1$, suppose we found the solution $(1, 3, 0, 2)$. At this point all hard binary constraints in the problem are satisfied, but z_0, z_1 and z_3 are not totally satisfying the respective supplementary fuzzy constraints. If the heuristic learning rule is now applied, the inputs of the label nodes $\langle z_0, 1\rangle$, $\langle z_2, 3\rangle$, $\langle z_3, 2\rangle$ will decrease from 0 to -0.9. Since an input of -0.9 is still the highest among all states, there will not be any variable that can change its state in the convergence cycle. If the heuristic learning rule is applied again, the inputs of the label nodes $\langle z_0, 1\rangle$, $\langle z_2, 3\rangle$ and $\langle z_3, 2\rangle$ will all decrease from -0.9 to -1.8. Depending on the preceding execution of fuzzy GENET(DCSP), it may now be possible for z_0, z_1

and z_3 to go to other states if their inputs are high enough. Note that z_1 will remain to be 0 because the fuzzy unary constraint "$z_1 = 0$" is not penalized. If fuzzy GENET(DCSP) continues to be stuck, the learning rule will be applied again until it eventually escapes the local maximum.

There are also 2 minor differences between the original and the new fuzzy GENET. First, as pointed out before, the algorithm will not stop once it is started and there is no way to know when an optimal solution is found, we need to make some predefined stopping criteria and monitor the program execution, once the stopping criteria are met we stop the execution (Algorithm 4.1, line 6). Stopping criteria can be any arbitrary limits, but they are usually resources bounds, such as the number of iterations, time budget, memory usage, distance upper bound, or any combinations of them.

Second, as we are using fuzzy GENET(DCSP) to solve DCSPs, we need a way to know how good the solutions are. Therefore a procedure for calculating the distance is required. This procedure is executed to find out the distance between the current and the preceding solutions when a local maximum is reached, and hence can be used to determine whether the current solution is better (Algorithm 4.1, line 19). If a better one is found, the solution is recorded for later comparison (Algorithm 4.1, line 20–21).

4.3 Experiments

Two sets of experiments are performed to evaluate the solution quality of fuzzy GENET(DCSP). They are presented in Sections 4.3.2 and 4.3.4 respectively. We test on randomly-generated problems and compare the solutions of fuzzy GENET(DCSP) against the optimal ones. Optimal solutions are obtained using a complete search method. All experiments were executed on Sun Ultra 5/270 workstations running Solaris 2.6.

The first set of experiment uses the Hamming distance (4.1) as the metric for

measuring stability, and the second set uses the Manhattan distance function (4.4). In each set, we varies the four parameters, number of variables, domain size, density and tightness, of the randomly-generated CSPs independently so that the performance can be observed individually. We adopt this notation to represent a randomly-generated CSP instance: x-y-z-w, which denotes a problem of constraint density $z\%$ and tightness $w\%$ having x variables, each having y domain values.

Besides varying the 4 parameters, we also study the effect on solution quality when different thresholds are used.

In order to make a fair comparison between solutions found by fuzzy GENET(DCSP) and optimal distances, we define a unit called *discrepancy*, which is a value larger than or equal to 0:

$$\text{discrepancy} = \frac{\text{distance} - \text{optimal distance}}{\text{optimal distance}} \tag{4.12}$$

Discrepancy is 0 when the solution found by fuzzy GENET(DCSP) is optimal.

Before presenting the empirical results and analysis, we first describe in the next section the DCSP instance generation procedure that we use in all of our experiments.

4.3.1 Dynamic CSP Generation

The DCSP instances tested in the experiments are constructed from randomly-generated CSPs. Since a DCSP is a sequence of CSPs, we *simulate* a DCSP by creating two different but consecutive (in the sense of a DCSP) CSPs, P_0 and P_1, out of a single CSP (Z, D, C). The steps involved are summarized as follows: given a CSP (Z, D, C), a subset of constraints C' is selected from C to make another CSP (Z, D, C'), while making sure that variables in Z are all involved in C'. A DCSP is thus created: (Z, D, C') as P_0 and (Z, D, C) as P_1. In this way, the constraints being asserted into P_0 are those in the set $C' - C$, making P_1 more restricted. Because we make sure that all variables in Z are

involved in C', the number of variables is unchanged.

In order to test on a wide variety of DCSP instances, the routine shown in Algorithm 4.2 is employed to generate a large set of different DCSPs from a single CSP. For testing purpose we define a DCSP instance as a triple $(P_1, C', \boldsymbol{v})$, where P_1 is the second CSP (Z, D, C) in the DCSP sequence. C' is the set of constraints in the preceding problem P_0. Vector $\boldsymbol{v}$ is the *starting point* for P_1. It is a vector of values denoting the variable assignment of a solution of P_0, which is also the initial variable assignment for P_1. Provided that P_0 is not an extremely tight CSP, there are many solutions for P_0, which will in turn lead to very different results for P_1. By regarding $\boldsymbol{v}$ as part of a DCSP instance, we introduce one more level of granularity of details for investigating the behaviours of our algorithms on solving different DCSPs.

The routine in Algorithm 4.2 requires 4 parameters:

P_1 : The second CSP (Z, D, C) in the DCSP sequence.

N_p : The number of different P_0's to be generated out of P_1. Since Z and D are the same for P_0 and P_1, N_p of different C' are created. For instance, if $N_p = 5$, then among all generated DCSP instances there will be 5 different P_0 (namely $(Z, D, C'_1), \ldots, (Z, D, C'_5)$).

N_{sp} : The number of different *starting points* for each P_0. That is, for each P_0, N_{sp} different starting points are selected.

R : This ratio of $|C'|$ to $|C|$.

The total number of DCSP instances generated by this routine is thus $N_p \times N_{sp}$.

In all of our experiments, we set N_p to 10 and N_{sp} to 10, and run the DCSP generation procedure 5 times with R set to 0.5, 0.6, 0.7, 0.8, and 0.9. Thus, for a CSP *x-y-z-w*, $10 \times 10 \times 5$, i.e. 500, DCSP instances are generated. Moreover, since fuzzy GENET(DCSP) is a stochastic algorithm, 10 runs are performed for

Procedure DCSP-generate (P_1, N_p, N_{sp}, R)
for $i \leftarrow 1$ to N_p **do**
  $DCPSs \leftarrow \emptyset$
  $C' \leftarrow$ randomly select $R\%$ of constraints from C
  (ensure all variables are involved in C')
  $P_0 \leftarrow (Z, D, C')$
  for $j \leftarrow 1$ to N_{sp} **do**
    solve P_0
    $S \leftarrow$ Solution of P_0
    $DCSPs \leftarrow DCSPs \cup (P_1, C', S)$
  end for
end for
return *DCSPs*

Algorithm 4.2: The DCSP generation routine

each DCSP instances in order to get an average of the results. In total, 5000 runs are performed for each x-y-z-w problem.

4.3.2 Problems Using Hamming Distance Function

The objective of this experiment is to test fuzzy GENET(DCSP) using DCSPs with Hamming distance (4.1) as the distance function, and evaluate the quality of solutions by comparing them with the optimal solutions. We used the Branch-and-Bound algorithm in ILOG Solver 4.0 [16, 15] to find the optimal solutions.

In each of the following subsections, we execute the experiments by varying one of the 4 randomly-generated CSP parameters. As the Hamming distance is being used, we employ the membership function (4.3), with $\alpha = 0.001$. The aggregate operator is minimum and the threshold is also 0.001.

As mentioned in Section 4.2, fuzzy GENET(DCSP) will only stop when some stopping criteria are met. In these experiments we use an iteration limit of 200,000 as the bound. Once the bound is reached the algorithm stops and the best distance found so far is reported.

4.3.2.1 Variation in Number of Variables

In this experiment, we compare problems of different number of variables: 40, 45 and 50. Results are shown in Table 4.4. The first column is the randomly-generated binary CSPs that correspond to P_1. The second column contains the $|C'| : |C|$ ratios. The third column is the average distance of all solutions found in the 1000 runs of fuzzy GENET(DCSP) for a single row. The fourth column contains the average optimal distances of the 100 DCSP instances for a single row. The fifth column shows the absolute difference of distances between the solutions found by fuzzy GENET(DCSP) (third column) and the optimal solutions (fourth column). The sixth column shows the percentage of discrepancy, which is calculated using (4.12).

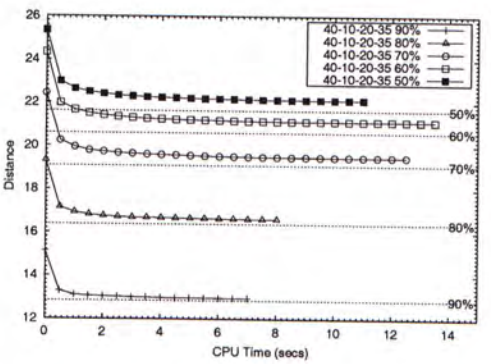
The seventh column shows the average number of iterations passed till the distance shown in the third column is reached. And the last two columns show the average CPU time spent on solving the problems, and they are mainly for reference only. For fuzzy GENET(DCSP), this is the time for the procedure to run until the stopping criteria are met. For the Branch-and-Bound method, it is the time that the procedure took to find the optimal solutions. We feed the best distance found by fuzzy GENET(DCSP) to the Branch-and-Bound algorithm as its initial bound. Its execution time will be longer if we do not do so. Thus it is meaningless to report its exact CPU time. Note that the time shown for the fuzzy GENET(DCSP) depends heavily on how the stopping criteria are set, since the algorithm will only terminate when the criteria are fulfilled. Suppose the stopping criteria is set to a very large iteration limit. It becomes unfair to compare the time taken by the fuzzy GENET(DCSP) and the Branch-and-Bound method, because fuzzy GENET(DCSP) may spend much of its time on looking for a better solution even after the optimum is reached.

In terms of solution quality, the fifth and sixth columns are what we are interested in. The differences of distance for all the problems we tested are between 0.101 and 1.319, which shows that solutions of fuzzy GENET(DCSP) do

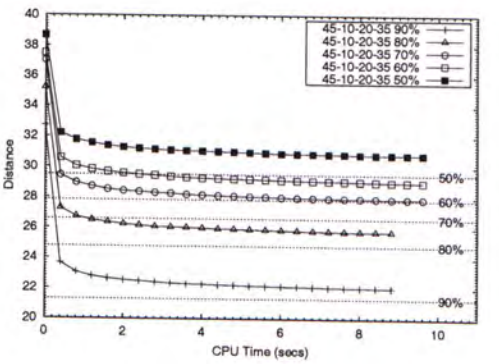
Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
40-10-20-35	90%	12.951	12.850	0.101	0.786	11433.1	0.49	> 200
	80%	16.615	16.390	0.225	1.373	19124.4	0.85	
	70%	19.468	19.090	0.378	1.980	25967.9	1.24	
	60%	21.121	20.600	0.521	2.529	28110.5	1.35	
	50%	22.124	21.620	0.504	2.331	27794.5	1.36	
45-10-20-35	90%	22.022	21.290	0.732	3.438	27859.7	1.38	> 1000
	80%	25.739	24.790	0.949	3.828	28272.6	1.46	
	70%	27.919	26.600	1.319	4.959	29013.5	1.51	
	60%	29.009	27.850	1.159	4.162	34211.2	1.77	
	50%	30.805	29.500	1.305	4.424	30954.0	1.64	
50-10-20-35	90%	36.863	35.990	0.873	2.426	28517.1	2.06	> 1000
	80%	39.091	38.270	0.821	2.145	29199.8	2.15	
	70%	40.868	40.070	0.798	1.992	30871.7	2.28	
	60%	40.936	40.240	0.696	1.730	26591.9	2.00	
	50%	41.404	40.600	0.804	1.980	26273.0	2.01	

Table 4.4: Comparison between fuzzy GENET(DCSP) and Branch-and-Bound on randomly-generated binary DCSPs of different number of variables, using Hamming distance

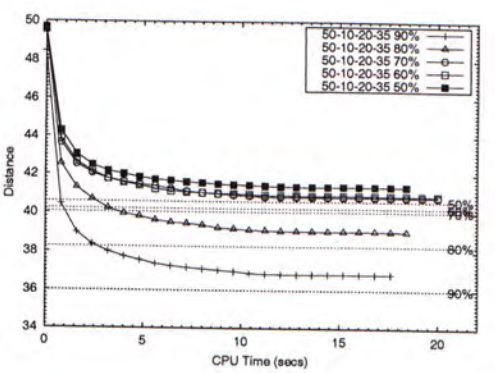
[a]Branch-and-Bound



(a) 40-10-20-35



(b) 45-10-20-35



(c) 50-10-20-35

Figure 4.2: Convergence behaviours of fuzzy GENET(DCSP) on problems with different number of variables when using the Hamming distance

not deviate from the optima more than 2 variables on average. The discrepancy for the 40-10-20-35 problem is quite small, which is below 3%. For the larger, and hence more difficult, problems, the discrepancy increases but is still below 5%.

As mentioned before, it is unfair to compare the time taken between the two algorithms. Nevertheless, fuzzy GENET(DCSP) still spends significantly less time than Branch-and-Bound. This confirms that for small to medium-sized problems, fuzzy GENET(DCSP) can attain near-optimal solutions in much shorter time.

Graphs in Figure 4.2 are plotted to visualize the convergence behaviours of

fuzzy GENET(DCSP) for all the problems tested in this section. Each curve shows the average convergence behaviour (1000 runs for 100 instances) of our method on instances of a particular $|C'| : |C|$ ratio in percentage. We also indicate the average optimal distance of each percentage as a horizontal dashed line. The plotting of each curve ends when all 1000 runs will not obtain any further improved solution after that time point. From the graphs it can be seen that fuzzy GENET(DCSP) converges very fast. For the 40- and 45-variables problems, it takes 1 to 2 seconds before the distance improvement slows down. Longer time is needed for the 50-variables problems as they are harder, but 10 seconds are already enough to let fuzzy GENET(DCSP) to converge. From these observations, we conclude that the convergence rate of fuzzy GENET(DCSP) is very good, which takes only a few seconds.

4.3.2.2 Variation in Domain Size

In the second experiment, we compare problems with domain sizes of 10, 15 and 20, with a fixed number of variables of 45, constraint density of 20%, and tightness of 35%. Results are shown in Table 4.5, which has the same layout as the last one. Some data in the branch-and-bound columns for the 45-15-20-35 and 45-20-20-35 problems are missing, since the branch-and-bound procedure takes much longer time to finish than on the 45-10-20-35 problems. It is not uncommon to have problems that spend more than one week to find the optimal solutions. It becomes more impractical to get the optimal distance when the problem size is larger. Nevertheless, results of fuzzy GENET(DCSP) are still reported (Table 4.5).

From the 90% rows in Table 4.5, we can see that fuzzy GENET(DCSP) also attains solutions that are close to the optimal ones. In fact, it is even more so when the domain size increases. This is not surprising as according to [35, 36], the expected number of solutions increases when the domain size grows[2], which

[2]For the problem m-n-p_1-p_2, the expected number of solutions E(N) is $m^n(1-$

Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
45-10-20-35	90%	22.022	21.290	0.732	3.438	27859.7	1.38	> 1000
	80%	25.739	24.790	0.949	3.828	28272.6	1.46	
	70%	27.919	26.600	1.319	4.959	29013.5	1.51	
	60%	29.009	27.850	1.159	4.162	34211.2	1.77	
	50%	30.805	29.500	1.305	4.424	30954.0	1.64	
45-15-20-35	90%	14.030	13.800	0.230	1.667	14654.5	1.27	9039.78
	80%	18.940	–	–	–	48754.3	3.88	–
	70%	21.500	–	–	–	42126.2	3.88	–
	60%	23.260	–	–	–	43057.5	4.48	–
	50%	25.260	–	–	–	36144.9	3.40	–
45-20-20-35	90%	12.330	12.200	0.130	1.066	15860.4	0.81	10039.40
	80%	16.710	–	–	–	32469.6	3.39	–
	70%	19.630	–	–	–	39594.8	4.18	–
	60%	20.380	–	–	–	44977.5	4.86	–
	50%	21.620	–	–	–	39870.0	4.50	–

Table 4.5: Comparison between fuzzy GENET(DCSP) and Branch-and-Bound on randomly-generated binary DCSPs of different domain sizes, using Hamming Distance

[a]Branch-and-Bound

makes it an easier stability problem.

4.3.2.3 Variation in Density and Tightness

The third experiment investigates the solution quality by altering the density and tightness of the randomly-generated binary CSPs while keeping the number of variables and domain sizes constant. It is difficult to find good testing values for the constraint density and tightness since, for problems of around 45 and 50 variables and domain size of 10, too high a value for the two parameters can easily result in unsatisfiable instances. Therefore, we fix the number of variables to 50 and domain size to 10, and test all four combinations of constraint density 15% and 20%, and constraint tightness 30% and 35%. Results are tabulated in Table 4.6. Since experiments on the 50-10-20-35 problems have already been done, they are copied from Table 4.4 for ease of comparison. Some results for the branch-and-bound procedure are not available since there is not enough time to finish them. The trend in Table 4.6 shows that fuzzy GENET(DCSP) exhibits better solution quality for the easier, *i.e.* less constrained, problems, and is generally faster at the same time.

4.3.3 Comparison in Using Different Thresholds

As it is unclear of the influence on the solution quality and execution time when different thresholds are used, we repeated some of the experiments in Section 4.3.2 and a 150-variables problem, with α and α_0 being set to 0.001, 0.01, 0.1, 0.5 and 0.9. The results are tabulated in 3 tables: Table 4.7 shows the comparison of solution quality, Table 4.8 compares the number of iterations used, and Table 4.9 compares the CPU time taken. From Table 4.7, we cannot find any evidence to confirm that there is a strong relationship between the threshold being used and the corresponding solution quality. The results do not show any trend on how the solution quality changes when different thresholds

$0.01 \cdot p_2)^{n(n-1)\cdot 0.01\cdot p_1/2}$.

Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
50-10-15-30	90%	9.410	9.400	0.010	0.106	13107.8	0.57	1826.26
	80%	13.420	–	–	–	22470.2	1.06	–
	70%	17.640	–	–	–	32174.7	1.61	–
	60%	18.820	–	–	–	28492.4	1.43	–
	50%	21.700	–	–	–	44952.7	2.24	–
50-10-15-35	90%	15.220	15.100	0.120	0.795	22108.2	1.01	3202.22
	80%	19.540	–	–	–	36386.1	1.66	–
	70%	22.880	–	–	–	35303.3	1.69	–
	60%	23.900	–	–	–	44931.1	2.21	–
	50%	27.200	–	–	–	37862.9	1.89	–
50-10-20-30	90%	15.890	15.700	0.190	1.210	18849.9	0.93	637.13
	80%	22.000	–	–	–	21757.2	1.15	–
	70%	25.920	–	–	–	35506.8	1.88	–
	60%	28.740	–	–	–	41856.2	2.27	–
	50%	30.120	–	–	–	43824.3	2.43	–
50-10-20-35	90%	36.863	35.990	0.873	2.426	28517.1	2.06	> 1000
	80%	39.091	38.270	0.821	2.145	29199.8	2.15	
	70%	40.868	40.070	0.798	1.992	30871.7	2.28	
	60%	40.936	40.240	0.696	1.730	26591.9	2.00	
	50%	41.404	40.600	0.804	1.980	26273.0	2.01	

Table 4.6: Comparison between fuzzy GENET(DCSP) and Branch-and-Bound on randomly-generated binary DCSPs of different constraint density and tightness, using Hamming distance

[a]Branch-and-Bound

Problem	$\|C'\| : \|C\|$	Distance found when threshold =				
		0.001	0.01	0.1	0.5	0.9
40-10-20-35	90%	12.947	12.937	12.971	12.936	12.980
	80%	16.603	16.609	16.590	16.666	16.587
	70%	19.517	19.494	19.471	19.473	19.463
	60%	21.092	21.098	21.106	21.105	21.105
	50%	22.142	22.101	22.130	22.101	22.093
50-10-20-35	90%	37.861	37.894	38.010	37.979	37.933
	80%	39.613	39.573	39.641	39.494	39.426
	70%	41.537	41.547	41.521	41.589	41.536
	60%	41.697	41.710	41.754	41.685	41.631
	50%	41.944	41.801	41.923	41.863	41.794
150-10-15-15	90%	60.858	61.045	61.079	61.785	61.650
	80%	80.717	80.970	80.689	81.269	81.345
	70%	89.280	89.408	89.507	89.870	89.990
	60%	95.597	95.536	95.400	95.803	96.022
	50%	99.437	99.351	99.516	99.964	99.552

Table 4.7: Comparison of solution quality on randomly-generated binary CSPs when different thresholds are used

Problem	$\|C'\| : \|C\|$	Number of iterations used when threshold =				
		0.001	0.01	0.1	0.5	0.9
40-10-20-35	90%	11064.6	11729.8	10076.6	11455.4	10269.8
	80%	19869.0	19582.6	19841.3	22290.2	19340.4
	70%	23371.0	23959.7	25434.7	23964.9	24527.0
	60%	27931.4	29296.6	28535.2	30988.4	27133.9
	50%	27256.4	28472.4	27061.7	28228.9	27021.8
50-10-20-35	90%	28436.3	27688.7	31442.7	32523.0	27527.8
	80%	30321.3	27000.6	30573.6	30350.8	27709.4
	70%	30694.1	31011.7	30451.3	31127.6	33208.0
	60%	31998.7	31452.3	32097.2	30661.8	31074.2
	50%	30486.7	28113.8	34116.9	30132.8	30282.4
150-10-15-15	90%	51229.8	48303.8	51654.2	53489.1	54343.2
	80%	47120.9	47961.0	50173.2	54049.6	58817.7
	70%	49225.5	47763.8	47976.0	51644.3	55759.4
	60%	48081.1	48132.6	48763.5	52741.1	55737.2
	50%	52076.0	53360.0	49966.4	54647.1	57090.7

Table 4.8: Comparison of the number of iterations used on randomly-generated binary CSPs when different thresholds are used

Problem	$\|C'\| : \|C\|$	CPU time (second) taken when threshold =				
		0.001	0.01	0.1	0.5	0.9
40-10-20-35	90%	0.49	0.58	0.44	0.49	0.46
	80%	0.88	0.98	0.87	0.96	0.86
	70%	1.05	1.22	1.12	1.06	1.10
	60%	1.27	1.50	1.29	1.37	1.23
	50%	1.26	1.47	1.23	1.28	1.24
50-10-20-35	90%	2.04	2.01	2.22	2.14	1.87
	80%	2.17	1.95	2.16	2.05	1.91
	70%	2.22	2.24	2.19	2.11	2.26
	60%	2.31	2.27	2.30	2.09	2.14
	50%	2.23	2.07	2.45	2.06	2.10
150-10-15-15	90%	10.32	9.65	10.36	10.70	10.98
	80%	10.02	10.12	10.65	11.31	12.34
	70%	10.71	10.40	10.52	11.13	12.12
	60%	10.77	10.71	10.84	11.51	12.31
	50%	11.70	11.94	11.23	12.19	12.81

Table 4.9: Comparison of CPU time taken on randomly-generated binary CSPs when different thresholds are used

are used. We conclude that the value of threshold does not affect the solution quality.

We also cannot find any relationship between the threshold being used and the efficiency of the algorithm. There does not exist a consistent change in the number of iterations and the CPU time with respect to the increase of the threshold.

Due to these observations, we find that which threshold value to use does not matter too much. Thus we stick to 0.001.

4.3.4 Problems Using Manhattan Distance Function

In this section, we evaluate the solutions quality of fuzzy GENET(DCSP) with the Manhattan distance function (4.4) as the distance metric. We define the membership function of the supplementary fuzzy constraints as in (4.5). The aggregate operator is minimum and the threshold is 0.1, because the lowest degree of satisfaction above 0 is $1 - \frac{9-0}{9-0+1}$, i.e. 0.1. Again, we use an iteration

limit of 200,000 as the stopping criterion.

As usual, all the optimal solutions are found by using the Branch-and-Bound algorithm in ILOG Solver 4.0 [16, 15]. Like the Hamming distance experiment, we vary the number of variables, domain size, and constraint density and tightness for comparison. The results are tabulated in Table 4.10, 4.11, and 4.12 respectively.

The difference of distance between solutions of fuzzy GENET(DCSP) and branch-and-bound is within 0.543 and 5.121 for all the problems. It is higher than those in the Hamming distance experiments, but given that the Manhattan distance is more demanding (the worst possible distances of the experiments are ranging from 360 to 855), and by the fact that the highest discrepancy is only 5.931%, fuzzy GENET(DCSP) still attains good solution quality in the Manhattan distance experiments. Although comparison with optimal solutions are unavailable in some of the experiments due to lack of branch-and-bound results, there does not seem to be abnormality in the outcome of the fuzzy GENET(DCSP) experiments. The figures in the three tables share the same trend as those in the Hamming distance experiments, and the analysis can be equally applied here. We conclude that fuzzy GENET(DCSP) is as effective (and efficient) in dealing with the Manhattan distance as in handling the Hamming distance problems.

As in the Hamming distance experiment, graphs for observing fuzzy GENET(DCSP)'s convergence behaviours are plotted in Figure 4.3. Only the problems on varying the domain size are plotted, since the others show similar trends. Curves of the 40- and 45-variables problems exhibit very fast convergence, while the 50-variables one show comparatively much slower convergence and irregularities in the curves. These both suggest that this problem is a very hard stability problem, and it may be necessary to lengthen the stopping criteria if more stable solutions are to be found.

Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
40-10-20-35	90%	37.213	36.670	0.543	1.481	21621.6	0.92	> 1500
	80%	46.697	45.370	1.327	2.925	31645.5	1.36	
	70%	52.718	50.840	1.878	3.694	37865.8	1.65	
	60%	57.778	55.370	2.408	4.349	42392.2	1.86	
	50%	58.400	56.420	1.980	3.509	40053.7	1.78	
45-10-20-35	90%	68.887	66.690	2.197	3.294	29024.8	1.43	> 1500
	80%	78.316	75.120	3.196	4.255	35643.7	1.80	
	70%	84.731	80.890	3.841	4.748	36431.2	1.85	
	60%	86.050	81.700	4.350	5.324	38087.8	1.94	
	50%	91.471	86.350	5.121	5.931	38192.2	1.95	
50-10-20-35	90%	128.807	124.520	4.287	3.443	28334.2	2.26	> 1500
	80%	133.286	129.980	3.306	2.543	27324.9	2.14	
	70%	138.222	134.910	3.312	2.455	29545.8	2.27	
	60%	142.517	139.070	3.447	2.479	30677.7	2.36	
	50%	142.822	139.150	3.672	2.639	28172.9	2.40	

Table 4.10: Comparison between fuzzy GENET(DCSP) and Branch-and-Bound on randomly-generated binary DCSPs of different number of variables, using Manhattan distance

[a]Branch-and-Bound

Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
45-10-20-35	90%	68.887	66.690	2.197	3.294	29024.8	1.43	> 1500
	80%	78.316	75.120	3.196	4.255	35643.7	1.80	
	70%	84.731	80.890	3.841	4.748	36431.2	1.85	
	60%	86.050	81.700	4.350	5.324	38087.8	1.94	
	50%	91.471	86.350	5.121	5.931	38192.2	1.95	
45-15-20-35	90%	55.410	–	–	–	46492.2	4.14	–
	80%	69.330	–	–	–	51224.1	4.75	–
	70%	73.230	–	–	–	50564.3	4.75	–
	60%	74.060	–	–	–	54243.0	5.16	–
	50%	82.030	–	–	–	63653.0	6.07	–
45-20-20-35	90%	51.790	–	–	–	44433.4	4.87	–
	80%	65.840	–	–	–	60695.2	6.78	–
	70%	73.190	–	–	–	53907.9	6.41	–
	60%	74.180	–	–	–	61429.2	7.33	–
	50%	81.900	–	–	–	68523.6	8.23	–

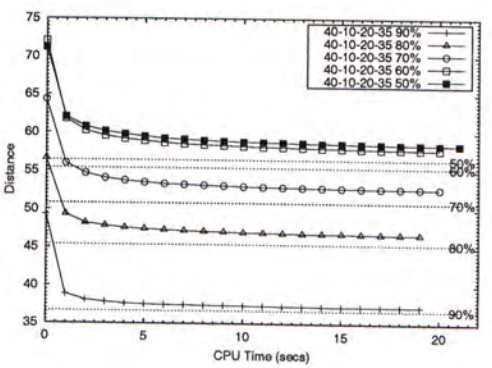
Table 4.11: Comparison between fuzzy GENET(DCSP) and Branch-and- Bound on randomly-generated binary DCSPs of different domain sizes, using Manhattan distance

[a]Branch-and-Bound

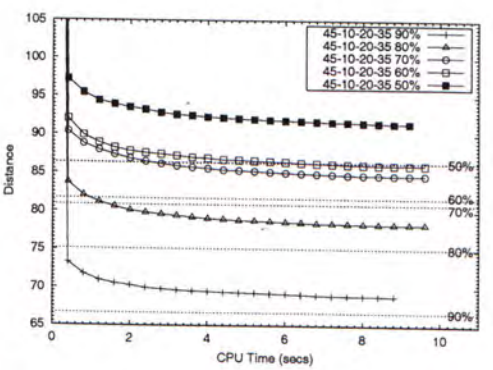
Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
50-10-15-30	90%	23.150	22.300	0.850	3.812	28955.9	1.39	72321.90
	80%	35.380	–	–	–	35261.9	1.92	–
	70%	39.820	–	–	–	49433.4	2.73	–
	60%	46.210	–	–	–	51809.2	2.94	–
	50%	49.740	–	–	–	52242.0	3.08	–
50-10-15-35	90%	41.270	39.800	1.470	3.693	32441.0	1.83	83756.40
	80%	53.980	–	–	–	48319.1	2.64	–
	70%	60.040	–	–	–	52126.8	2.93	–
	60%	61.800	–	–	–	48898.7	2.83	–
	50%	69.570	–	–	–	54844.0	3.20	–
50-10-20-30	90%	49.260	47.600	1.660	3.487	38355.3	2.42	8911.49
	80%	63.780	–	–	–	47376.8	3.07	–
	70%	71.000	–	–	–	37370.7	2.59	–
	60%	81.300	–	–	–	42111.0	2.96	–
	50%	83.140	–	–	–	48677.3	3.41	–
50-10-20-35	90%	128.807	124.520	4.287	3.443	28334.2	2.26	
	80%	133.286	129.980	3.306	2.543	27324.9	2.14	
	70%	138.222	134.910	3.312	2.455	29545.8	2.27	> 1500
	60%	142.517	139.070	3.447	2.479	30677.7	2.36	
	50%	142.822	139.150	3.672	2.639	28172.9	2.40	

Table 4.12: Comparison between fuzzy GENET(DCSP) and Branch-and-Bound on randomly-generated binary DCSPs of different constraint density and tightness, using Manhattan distance

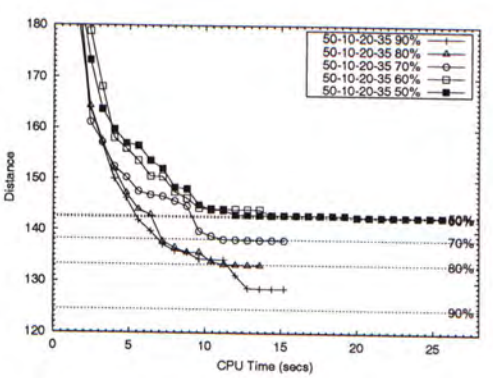
[a]Branch-and-Bound



(a) 40-10-20-35



(b) 45-10-20-35



(c) 50-10-20-35

Figure 4.3: Convergence behaviours of fuzzy GENET(DCSP) on problems with different number of variables when using the Manhattan distance

Chapter 5

Enhancement of the Modelling Scheme

In this chapter, we propose an enhancement to the modelling scheme and the fuzzy GENET(DCSP) procedure that we discussed in the last chapter. In the new scheme, we use a fuzzy constraint called *distance bound* for any stability problems. Employing distance bound in our new convergence procedure, fuzzy GENET(DCSP_2), which is based on fuzzy GENET(DCSP) and is designed for the distance bound, we can limit the space being searched and achieve better solutions within the same time limit. Since the distance bound is an n-ary constraint, fuzzy GENET(DCSP_2) is designed to be able to handle it.

5.1 Distance Bound

In our original scheme, what kind of fuzzy constraints to use depends on how the distance is defined in the stability problem. Modelling the problem may require some efforts if the distance requirement is unusual. In the new scheme, this is simplified to using only one supplementary fuzzy constraint called the *distance bound*. Given a stability problem $(\boldsymbol{s}^i, P_i, P_{i+1}, d)$, the distance bound

takes the following form:

$$d(\boldsymbol{z}, \boldsymbol{s}^i) = 0 \tag{5.1}$$

where $\boldsymbol{z} = (z_1, \ldots, z_n)$ is the tuple of all variables of P_{i+1} (and also P_i). The FCSP must use the minimum function as the aggregate operator in this case.

The ideas behind the distance bound and the supplementary fuzzy constraints used in our last scheme are the same: exploit the degrees of satisfaction to express preferences for the tuples. Initially, degrees of satisfaction of the tuples in the distance bound constraint are assigned in a monotonically decreasing fashion from variable assignments that give a distance of zero (the best solutions) to those that give the largest distance (the worst solutions). The suggested initialization for variable assignment $\boldsymbol{z}$ is:

$$\mu_{R^f_{\text{dist}}}(\boldsymbol{a}) = \alpha_0 + (1 - \alpha_0)\,\frac{d_{MAX} - d(\boldsymbol{a}, \boldsymbol{s}^i)}{d_{MAX}} \tag{5.2}$$

where α_0 is the threshold and d_{MAX} is the largest possible distance. This equation makes equal increment of degree of satisfaction from the worst distance to the best distance, and assigns the value of threshold as the degree of satisfaction for the worst distance and 1.0 to that of the best distance.

For instance, if there are 10 variables in the problem with the threshold of 0.01, and Hamming distance is used as the distance function, according to (5.2), degrees of satisfaction of all possible variable assignments can be initialized like this:

$d(\boldsymbol{a}, \boldsymbol{s}^i)$	0	1	2	3	4	5	6	7	8	9	10
$\mu_{r^f_{\text{dist}}}(\boldsymbol{a})$	1.000	0.901	0.802	0.703	0.604	0.505	0.408	0.307	0.208	0.109	0.010

5.2 Enhancement of Convergence Procedure

If fuzzy GENET(DCSP) is used, a forceful penalization will be done on the distance bound when a solution is found. However, we perceive that there are two enhancements that can be made to the distance bound constraint to boost the solution quality. The first one is to treat all variable assignments of the same

distance as one entity, such that there is only one degree of satisfaction for each distance, even though there are in fact many different variable assignments that are of the same distance, and the second one is to replace the forceful penalization step to lowering the degrees of satisfaction of all poorer solutions to zero.

The first enhancement is to treat variable assignments of the same distance as one entity. Since the distance bound involves all variables of the problem, the total number of tuples is $\Pi_{z \in Z} D_z$. The storage needed for keeping this number of weights can be enormous. For a problem with 10 variables, each variable having 10 values, the total number of tuples is 10^{10}. Suppose it takes 4 bytes to store the weight, 4×10^{10} bytes are required to store all of them, that is about 40 GBytes of memory, which is immense in today's computers. Although the memory consumption can dramatically be reduced by storing only the weights that have been decremented due to penalization as proposed by Lee *et al.* [19], weights are stored in an AVL-tree as suggested, which induces overhead when a new weight has to be added. More importantly, we conceive that there is no reason to store the weights of the tuples with the same distance individually. It makes more sense to treat all the tuples of the same distance equally by sharing a single weight, because they are equally that bad in the sense of stability problems. In this way we can also unify the heuristic learning of these tuples. Besides, it can simplify the implementation, reduce memory consumption, and shorten the execution time because indexing the weight can be done in $O(1)$ by merely using an array.

Furthermore, fuzzy GENET(DCSP) is unable to handle the n-ary distance bound because connections in its network cannot relate more than two variables. In order to implement the new idea and to handle the n-ary distance bound constraint, we adapt the idea of *constraint node* and *intermediate node* from E-GENET [18, 48, 19] to our new convergence procedure. Basically this is done

by breaking the distance bound into two constraints:

$$D = d(\boldsymbol{z}, \boldsymbol{s}^i) \wedge D = 0 \tag{5.3}$$

where D is an artificial variable specially created for the transformation, and its domain is the set of all possible distance between $\boldsymbol{z}$ and $\boldsymbol{s}^i$ calculated using the distance metric d. Fuzzy GENET(DCSP) does not handle the crisp $(n+1)$-ary constraint $D = d(\boldsymbol{z}, \boldsymbol{s}^i)$ as a regular constraint. It is used purely to determine the value for D according to the current variable assignment $\boldsymbol{z}$. Thus there is no degree of satisfaction and weight associated with the tuples in $D = d(\boldsymbol{z}, \boldsymbol{s}^i)$. Instead $D = 0$ is the actual fuzzy constraint that fuzzy GENET(DCSP) deals with like other normal constraints. It guides the search according to the user's preference for solutions, which is recorded in the degrees of satisfaction of tuples in this unary fuzzy constraint.

The second enhancement is to make the search concentrate on finding more stable solutions. Once a better solution, say $\boldsymbol{s}_0^{i+1}$, is found, the degrees of satisfaction of all tuples $\boldsymbol{a}$ in $D = 0$ such that $d(\boldsymbol{a}, \boldsymbol{s}^i) \geq d(\boldsymbol{s}_0^{i+1}, \boldsymbol{s}^i)$ are dropped to zero immediately, while keeping the others unaffected. Moreover, among such tuples, weights of those that are between 0 and -1 will be lowered to -1 at the same time. Weights that have been cumulated to below -1 are retained. Using the example in the last section, if $d(\boldsymbol{s}_0^{i+1}, \boldsymbol{s}^i) = 5$, the degrees of satisfaction of the tuples in the distance bound will become:

$d(\boldsymbol{a}, \boldsymbol{s}^i)$	0	1	2	3	4	5	6	7	8	9	10
$\mu_{R^f_{\text{dist}}}(\boldsymbol{a})$	1.000	0.901	0.802	0.703	0.604	0.000	0.000	0.000	0.000	0.000	0.000

Recall that the objective of the forceful penalization step in fuzzy GENET(DCSP) is to make the search depart from the local maximum so that the search can continue to other states. The above method can also achieve this purpose, because the degree of satisfaction of the current variable assignment has become zero, and thus the global degree of satisfaction is now below the threshold. This makes the current state unacceptable, and the search will then look for other better states.

Besides that, the main advantage of this method is it can effectively avoid many regions in the search spaces that do not have better solutions. Because their degrees of satisfaction are now lower than the threshold, weights will gradually grow up on these tuples as the search progresses. This will build up a significant factor over time to deter the search from going into the worse regions again, and thus will concentrate in the better areas. This can speed up the search and we expect it to achieve better results within the same time budget or iteration limit than using fuzzy GENET(DCSP) with the original modelling scheme.

Combining these ideas with fuzzy GENET(DCSP), we devise our new convergence procedure fuzzy GENET(DCSP_2) in Algorithm 5.1. The major change to Algorithm 4.1 is the forceful penalization in lines 19–23 has been replaced by resetting of degrees of satisfaction and weights in lines 22–28 of Algorithm 5.1.

5.3 Comparison with Optimal Solutions

The first set of experiments is to compare the solutions of fuzzy GENET(DCSP_2) to optimal solutions, and study how good the new method performs in terms of solution quality as well as efficiency. Except the membership function is now Equation. 5.2, all the other experimental settings are the same as those used in the last chapter's experiments: the aggregate operator is the minimum function, the threshold for the Hamming distance experiments is 0.001 and that for the Manhattan distance experiments is 0.1, and the stopping criterion is 200,000 iterations. Problems tested are 40-10-20-35, 45-10-20-35 and 50-10-20-35, Results are tabulated in Table 5.1 for the Hamming distance experiment and Table 5.2 for the Manhattan distance one.

Solution qualities on both sets of experiments are very satisfactory. Generally, the trend is that it becomes more difficult to maintain good solution quality when the CSP problems become harder. For the Hamming distance

1: initialize states of nodes as the last solution
2: **for** each $W_{\langle i,j\rangle\langle k,l\rangle}$ **do**
3: $\quad W_{\langle i,j\rangle\langle k,l\rangle} \leftarrow \alpha_{\langle i,j\rangle\langle k,l\rangle} - 1$
4: **end for**
5: **for** each $W_{d(z,z^0)}$ **do**
6: $\quad W_{d(z,z^0)} \leftarrow \alpha_{d(z,z^0)} - 1$
7: **end for**
8: $d \leftarrow$ max distance $+ 1$
9: **while** stopping criteria not met and $d \neq 0$ **do**
10: $\quad$ **while** $\alpha_P < \alpha_0$ **do**
11: $\quad\quad$ **for** each cluster **do**
12: $\quad\quad\quad$ *(asynchronously in parallel)*
13: $\quad\quad\quad$ calculate the input of each label node
14: $\quad\quad\quad$ turn on the node with maximum input
15: $\quad\quad$ **end for**
16: $\quad\quad$ **if** local maximum reached **then**
17: $\quad\quad\quad$ **for** each $W_{\langle i,j\rangle\langle k,l\rangle}$ that contributes to the FCSP violation **do**
18: $\quad\quad\quad\quad$ update $W_{\langle i,j\rangle\langle k,l\rangle}$: $W^{new}_{\langle i,j\rangle\langle k,l\rangle} = W^{old}_{\langle i,j\rangle\langle k,l\rangle} + O^{old}_{\langle i,j\rangle} O^{old}_{\langle k,l\rangle} (\alpha_{\langle i,j\rangle\langle k,l\rangle} - 1)$
19: $\quad\quad\quad$ **end for**
20: $\quad\quad$ **end if**
21: $\quad$ **end while**
22: $\quad$ record solution: $z^0 \leftarrow z$
23: $\quad d \leftarrow$ current distance
24: $\quad$ {*update distance bound*}
25: $\quad$ **for all** $d(\boldsymbol{z}, \boldsymbol{z^0}) >= d$ **do**
26: $\quad\quad \alpha_{d(z,z^0)} \leftarrow 0$
27: $\quad\quad W_{d(z,z^0)} \leftarrow -1$
28: $\quad$ **end for**
29: **end while**

Algorithm 5.1: The fuzzy GENET(DCSP$_2$) convergence procedure

Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	Avg CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
40-10-20-35	90%	12.852	12.850	0.002	0.007	850.2	0.09	> 200
	80%	16.398	16.390	0.008	0.034	1432.1	0.13	
	70%	19.099	19.090	0.009	0.043	2653.9	0.24	
	60%	20.607	20.600	0.007	0.036	2826.9	0.26	
	50%	21.637	21.620	0.017	0.092	3240.2	0.30	
45-10-20-35	90%	21.307	21.290	0.017	0.072	5279.0	0.55	> 1000
	80%	24.847	24.790	0.057	0.282	9752.0	1.01	
	70%	26.635	26.600	0.035	0.190	9227.1	0.96	
	60%	27.893	27.850	0.043	0.251	9232.2	0.97	
	50%	29.565	29.500	0.065	0.419	10567.9	1.17	
50-10-20-35	90%	36.401	35.990	0.411	2.934	22816.4	2.64	> 1000
	80%	38.627	38.270	0.357	3.043	23264.9	2.71	
	70%	40.362	40.070	0.292	2.941	23000.4	2.68	
	60%	40.607	40.240	0.367	3.760	23895.2	2.76	
	50%	40.924	40.600	0.324	3.447	24941.5	2.87	

Table 5.1: Comparison between fuzzy GENET(DCSP_2) and Branch-and-bound on randomly-generated binary DCSPs of different number of variables, using Hamming distance

[a]Branch-and-Bound

Problem	$\|C'\| : \|C\|$	Distance	Optimal Distance	Diff. of Distance	Discrepancy (%)	No. of Iterations	Avg CPU time (seconds)	
							fuzzy GENET(DCSP)	BB[a]
40-10-20-35	90%	36.685	36.670	0.015	0.005	7454.7	0.54	> 1500
	80%	45.436	45.370	0.066	0.021	12001.7	0.84	
	70%	50.959	50.840	0.119	0.038	14167.6	0.98	
	60%	55.497	55.370	0.127	0.042	18122.8	1.21	
	50%	56.552	56.420	0.132	0.043	18478.5	1.26	
45-10-20-35	90%	67.007	66.690	0.317	0.094	21265.6	1.39	> 5000
	80%	75.597	75.120	0.477	0.145	24738.3	1.62	
	70%	81.444	80.890	0.554	0.171	30544.3	1.93	
	60%	82.240	81.700	0.540	0.167	28839.4	1.80	
	50%	87.140	86.350	0.790	0.248	29451.3	1.82	
50-10-20-35	90%	127.773	124.520	3.253	0.999	27431.9	1.40	> 1500
	80%	133.024	129.980	3.044	0.951	27860.5	1.43	
	70%	137.546	134.910	2.636	0.837	29099.2	1.49	
	60%	141.928	139.070	2.858	0.919	27538.0	1.40	
	50%	141.777	139.150	2.627	0.845	30319.3	1.51	

Table 5.2: Comparison between fuzzy GENET(DCSP_2) and Branch-and-bound on randomly-generated binary DCSPs of different number of variables, using Manhattan distance

[a]Branch-and-Bound

experiments, the absolute differences of distances of the 40- and 45-variables problems are well below 0.1, and that of 50-10-20-35 problem is below 0.5, showing that results are very near to optimal solutions. Results on the Manhattan distance experiments have similar trends, but the results of the 50-10-20-35 problem are distinctly further away from the optima, indicating that it is a rather hard stability problem.

5.4 Comparison with Fuzzy GENET(DCSP)

In this section fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) are compared. As both methods are much faster than the branch-and-bound algorithm, it becomes practical to examine them on larger problems. We have chosen problems with 150 variables and domain size from 10 to 50. As we already obtained the results of both of them on the smaller problems from previous experiments, we will first compare them by using these data directly.

5.4.1 Medium-sized Problems

Data obtained from experiments in Section 4.3 and 5.3 are reused to compare fuzzy GENET(DCSP_2) with fuzzy GENET(DCSP). For ease of comparison, results of the two methods are put together into Table 5.3 and 5.4, with a new column "Distance difference" added, which is calculated by subtracting the distance of fuzzy GENET(DCSP_2) from the distance of fuzzy GENET(DCSP). Apparently, by looking at this column, solutions of fuzzy GENET(DCSP_2) are more stable than those of fuzzy GENET(DCSP). The difference is not large since fuzzy GENET(DCSP) already accomplishes near-optimal results and leaves little room for improvement.

Problem	$\|C'\| : \|C\|$	fuzzy GENET(DCSP)			fuzzy GENET($DCSP_2$)			Distance difference
		Distance	Iterations	Time (s)	Distance	Iterations	Time (s)	
40-10-20-35	90%	12.951	11433.1	0.49	12.852	850.2	0.09	0.099
	80%	16.615	19124.4	0.85	16.398	1432.1	0.13	0.217
	70%	19.468	25967.9	1.24	19.099	2653.9	0.24	0.369
	60%	21.121	28110.5	1.35	20.607	2826.9	0.26	0.514
	50%	22.124	27794.5	1.36	21.637	3240.2	0.30	0.487
45-10-20-35	90%	22.022	27859.7	1.38	21.307	5279.0	0.55	0.715
	80%	25.739	28272.6	1.46	24.847	9752.0	1.01	0.892
	70%	27.919	29013.5	1.51	26.635	9227.1	0.96	1.284
	60%	29.009	34211.2	1.77	27.893	9232.2	0.97	1.116
	50%	30.805	30954.0	1.64	29.565	10567.9	1.17	1.240
50-10-20-35	90%	36.863	28517.1	2.06	36.401	22816.4	2.64	0.462
	80%	39.091	29199.8	2.15	38.627	23264.9	2.71	0.464
	70%	40.868	30871.7	2.28	40.362	23000.4	2.68	0.506
	60%	40.936	26591.9	2.00	40.607	23895.2	2.76	0.329
	50%	41.404	26273.0	2.01	40.924	24941.5	2.87	0.480

Table 5.3: Comparison between fuzzy GENET(DCSP) and fuzzy GENET($DCSP_2$) on randomly-generated binary DCSPs of different number of variables, using Hamming distance

Problem	$\lvert C' \rvert : \lvert C \rvert$	fuzzy GENET(DCSP)			fuzzy GENET($DCSP_2$)			Distance difference
		Distance	Iterations	Time (s)	Distance	Iterations	Time (s)	
40-10-20-35	90%	37.213	21621.6	0.92	36.685	7454.7	0.54	0.528
	80%	46.697	31645.5	1.36	45.436	12001.7	0.84	1.261
	70%	52.718	37865.8	1.65	50.959	14167.6	0.98	1.759
	60%	57.778	42392.2	1.86	55.497	18122.8	1.21	2.281
	50%	58.400	40053.7	1.78	56.552	18478.5	1.26	1.848
45-10-20-35	90%	68.887	29024.8	1.43	67.007	21265.6	1.39	1.880
	80%	78.316	35643.7	1.80	75.597	24738.3	1.62	2.719
	70%	84.731	36431.2	1.85	81.444	30544.3	1.93	3.287
	60%	86.050	38087.8	1.94	82.240	28839.4	1.80	3.810
	50%	91.471	38192.2	1.95	87.140	29451.3	1.82	4.331
50-10-20-35	90%	128.807	28334.2	2.26	127.773	27431.9	1.40	1.034
	80%	133.286	27324.9	2.14	133.024	27860.5	1.43	0.262
	70%	138.222	29545.8	2.27	137.546	29099.2	1.49	0.676
	60%	142.517	30677.7	2.36	141.928	27538.0	1.40	0.589
	50%	142.822	28172.9	2.40	141.777	30319.3	1.51	1.045

Table 5.4: Comparison between fuzzy GENET(DCSP) and fuzzy GENET($DCSP_2$) on randomly-generated binary DCSPs of different number of variables, using Manhattan distance

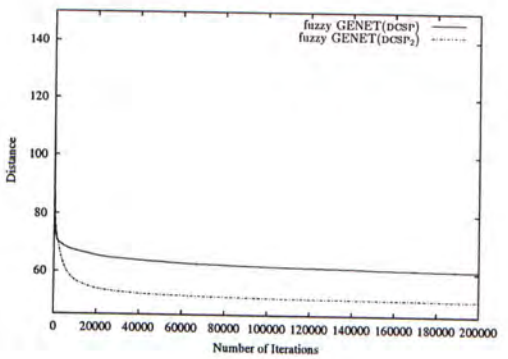
$\|C'\|/\|C\|$	fuzzy GENET(DCSP)			fuzzy GENET($DCSP_2$)			Dist.
	Dist.	Iteration	Time(s)	Dist.	Iteration	Time(s)	diff.
90%	60.777	51545.3	10.28	50.310	64831.1	27.33	10.467
80%	80.887	46547.0	9.89	66.237	72062.6	30.70	14.650
70%	89.547	49313.8	10.71	74.519	74839.0	32.03	15.028
60%	95.490	52806.6	11.68	80.729	74888.2	32.46	14.761
50%	99.483	48223.2	10.88	85.078	73395.2	32.01	14.405

Table 5.5: Comparison between fuzzy GENET(DCSP) and fuzzy GENET($DCSP_2$) on the 150-10-15-15 randomly-generated binary DCSPs, using Hamming Distance

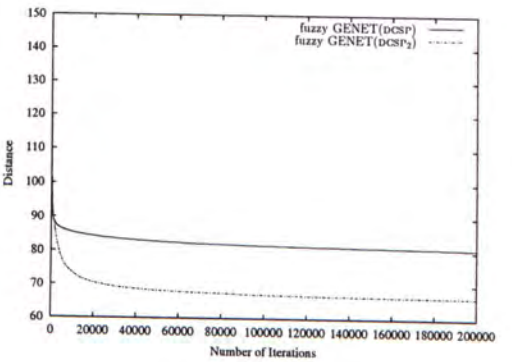
5.4.2 The 150-10-15-15 Problem

The objective of this experiment is to compare fuzzy GENET($DCSP_2$) with fuzzy GENET(DCSP) only. There are no optimal solutions for comparison since the time needed to find them will be exceptionally long. Due to the large number of variables, the problem's density and tightness have to be adjusted accordingly to 15 and 15 respectively, as unsolvable problems begin to appear when higher density and tightness are used. Except the problem size, all other experimental settings are identical to the ones used in Section 5.3. As usual, we test both Hamming and Manhattan distance functions.

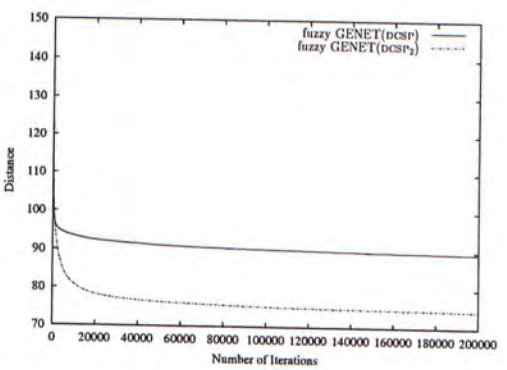
Results for Hamming distance are tabulated in Table 5.5. Distances of solutions of fuzzy GENET($DCSP_2$) are generally about 10 to 15 better than those of fuzzy GENET(DCSP), showing that fuzzy GENET($DCSP_2$) can achieve even more stable solutions on larger problems. Data in Table 5.5 shows that fuzzy GENET($DCSP_2$) seemingly takes longer time and more iterations to finish. However, these figures are actually the averaged value of time and number of iterations used when the best solutions are found. To conduct a more comprehensive study, we plot two sets of curves (Figures 5.1 and 5.2) to observe the change of solution stability when the program executes. The first set of graphs plots the change of distance as the number of iteration increases, and the second set plots the change of distance as time progresses. The latter one is needed because the time elapsed for one iteration in fuzzy GENET(DCSP) and in fuzzy



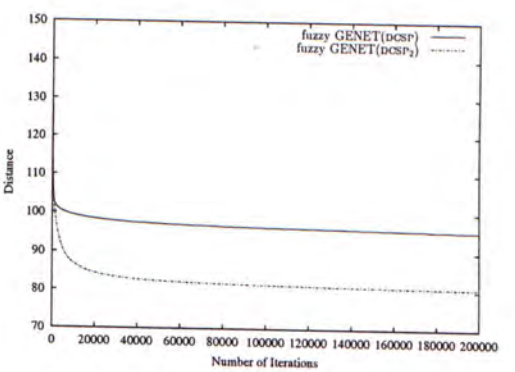
(a) 90%



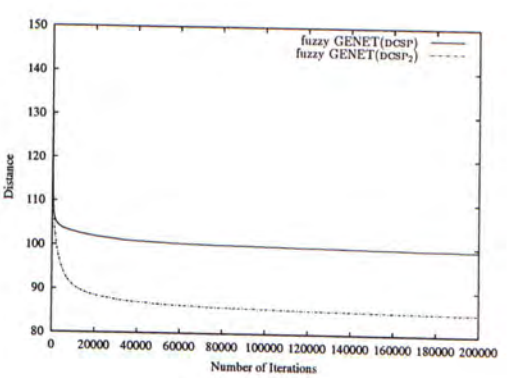
(b) 80%



(c) 70%

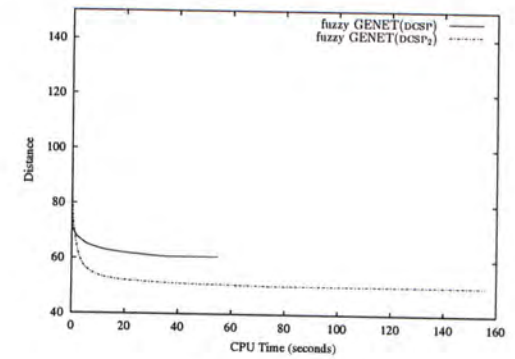


(d) 60%

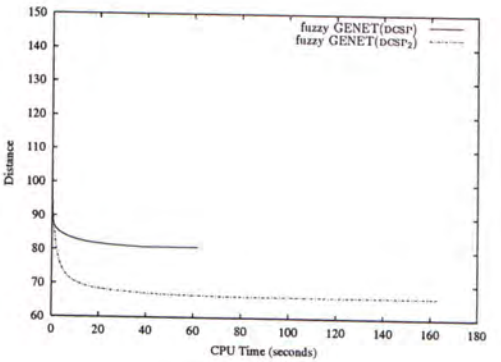


(e) 50%

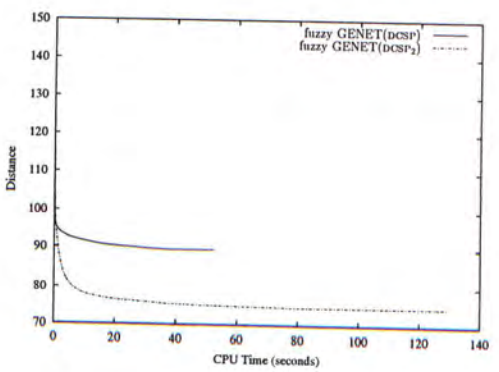
Figure 5.1: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-15-15 problem, using Hamming distance



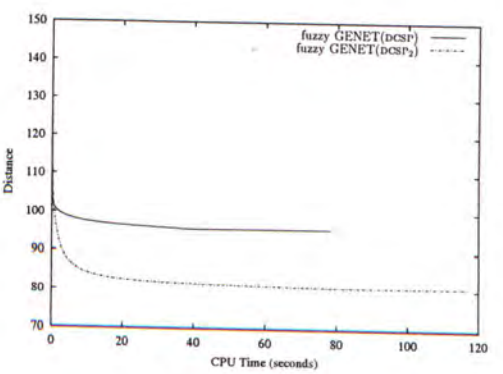
(a) 90%



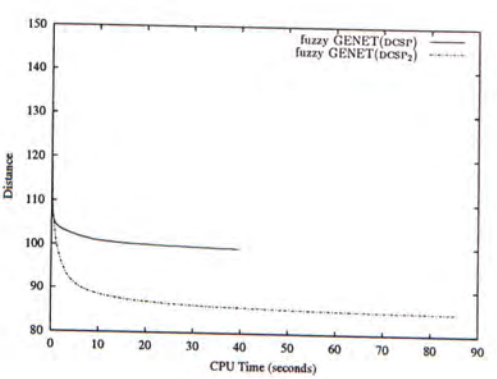
(b) 80%



(c) 70%



(d) 60%



(e) 50%

Figure 5.2: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-15-15 problem, using Hamming distance

$\lvert C' \rvert / \lvert C \rvert$	fuzzy GENET(DCSP)			fuzzy GENET($DCSP_2$)			Dist.
	Dist.	Iteration	Time(s)	Dist.	Iteration	Time(s)	diff.
90%	199.446	58517.8	11.52	172.563	105308.0	39.59	26.883
80%	251.351	53938.3	11.16	213.461	106708.0	38.90	37.890
70%	271.385	47922.4	10.26	231.457	104906.0	37.76	39.928
60%	285.902	49348.0	10.74	244.453	102233.0	36.59	41.449
50%	293.162	51333.0	11.21	251.174	103514.0	41.01	41.988

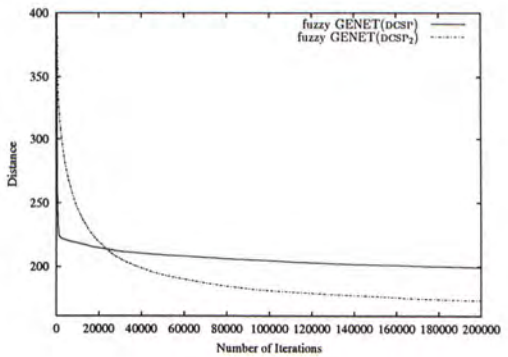
Table 5.6: Comparison between fuzzy GENET(DCSP) and fuzzy GENET($DCSP_2$) on the 150-10-15-15 randomly-generated binary DCSPs, using Manhattan distance

GENET($DCSP_2$) are different. Please note that the CPU time corresponding to a curve's end point is the time for the last improvement among all the runs. For example in Figure 5.2(a), the fuzzy GENET(DCSP) curve ends at 54.84 seconds, which means there is a run that makes its last improvement at 54.84 second, and there is no other runs that have made any improvement since then. It has no relationship to the value of time shown in the table (which is the average CPU time spent), therefore they are different.

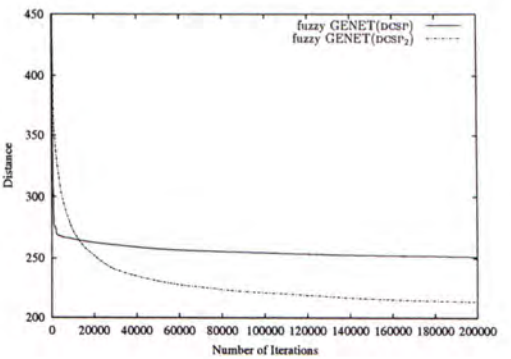
Figures 5.1 and 5.2 show that fuzzy GENET($DCSP_2$) begins to attain better results than fuzzy GENET(DCSP) at the very early moment of program execution, and it continues to be so throughout the whole execution. That means the overall performance of fuzzy GENET($DCSP_2$) is superior.

Results for Manhattan distance experiments are tabulated in Table 5.6. Once again fuzzy GENET($DCSP_2$) exhibits similar behaviour as in the last Hamming distance experiments: more stable solutions as distance difference is from 26.883 to 41.988, but it seemingly uses more iterations and longer time. The same kind of curves are again plotted in Figures 5.3 and 5.4 to study the change of distance as the program executes.

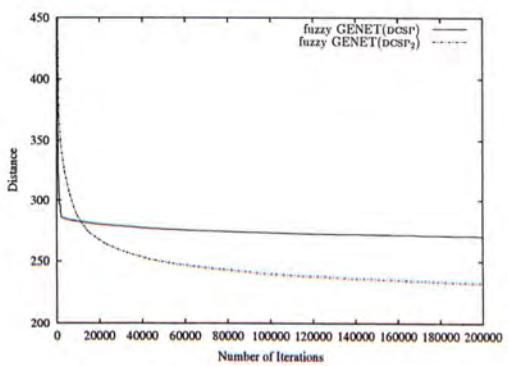
The graphs display a special characteristic: fuzzy GENET(DCSP) attains a relatively good distance rapidly at the very beginning, and after that the distance improvement slows down. On the other hand, fuzzy GENET($DCSP_2$) does not do as good as fuzzy GENET(DCSP) at the beginning (before the first 20000



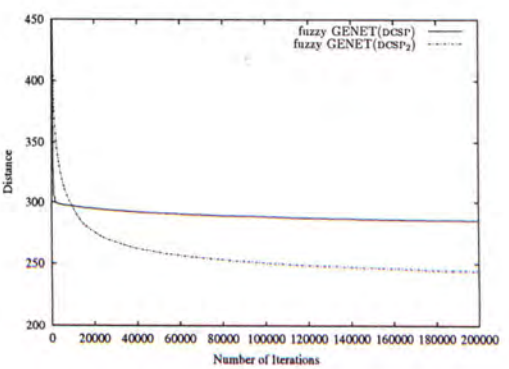
(a) 90%



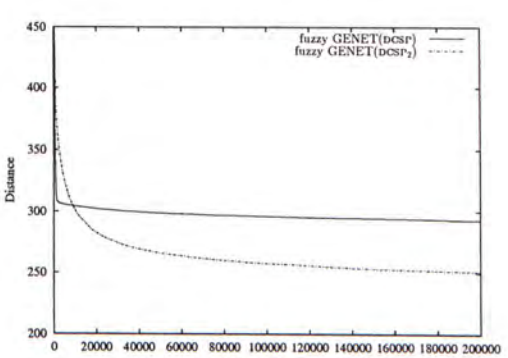
(b) 80%



(c) 70%

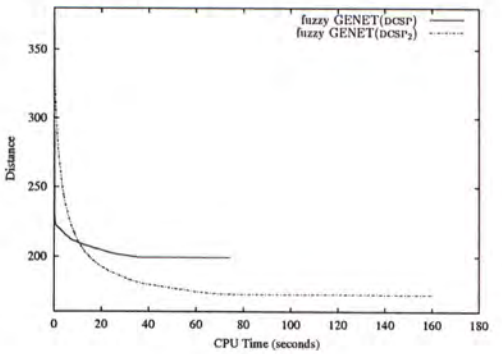


(d) 60%

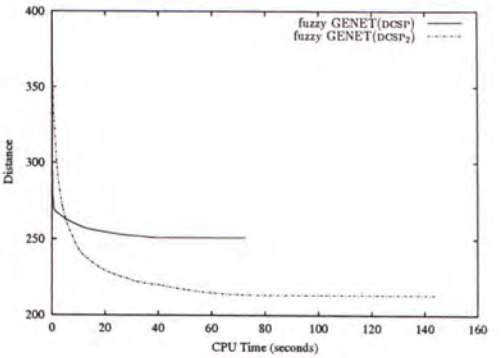


(e) 50%

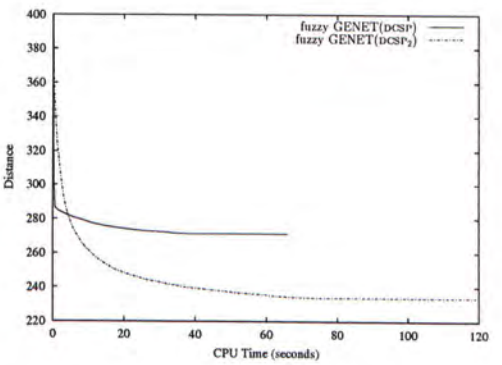
Figure 5.3: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-15-15 problem, using Manhattan distance



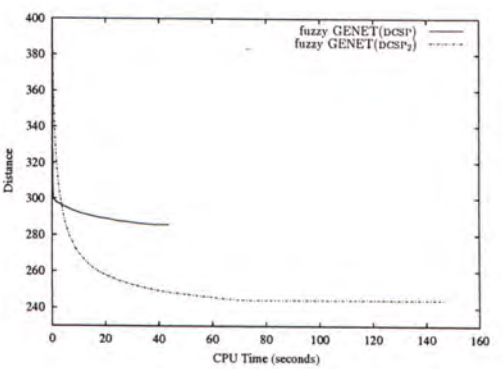
(a) 90%



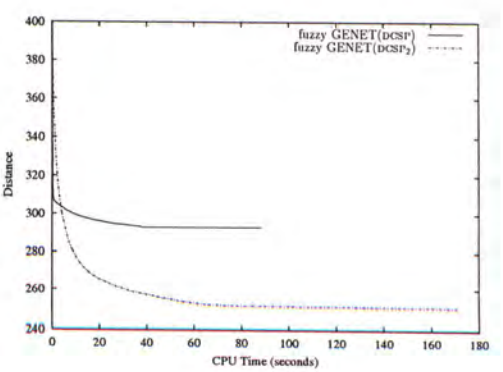
(b) 80%



(c) 70%



(d) 60%



(e) 50%

Figure 5.4: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-15-15 problem, using Manhattan distance

iterations or 10 seconds), but the distance keeps going down until it finally passes the best distance of fuzzy GENET(DCSP) and achieves much better results at the end. We delay the discussion of this phenomenon to Section 5.5, because some of the results in subsequent experiments also have the same phenomenon. Nevertheless, fuzzy GENET(DCSP_2) outperforms fuzzy GENET(DCSP) in the long run.

5.4.3 Variation in Density and Tightness

The next 3 sets of experiments again test problems of 150 variables and domain size of 10. By varying the constraint density and tightness, we have a diversity of problems with different numbers of constraints and invalid tuples per variable. Since we cannot raise the values of density and tightness any more (over-constrained problems will appear), we elect to lower both of them to 10, which makes up to 3 combinations of constraint density and tightness: 10-10, 10-15 and 15-10. Apart from the change in the numbers of constraints and invalid tuples, the numbers of possible solutions are also different as they vary. According to Smith and Dyer [35, 36], the expected number of solutions in these problems decreases in this order: 150-10-10-10 > 150-10-15-10 > 150-10-10-15 > 150-10-15-15.

The best distance obtained and the distance difference are tabulated in Table 5.7 for Hamming distance and Table 5.8 for Manhattan distance experiments. For all the Hamming distance experiments, distances of both methods are very close, which illustrates that they perform equally well in the less constrained CSPs. However, fuzzy GENET(DCSP_2) evidently achieves better solutions in the Manhattan distance experiments. We can see that the distance function plays a crucial role in testing the effectiveness of a stability algorithm. Given the same CSP, although fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) performs equally well in the Hamming distance experiment, fuzzy GENET(DCSP_2) is capable of finding better solutions when the more de-

Problem	$\|C'\| : \|C\|$	fuzzy GENET(DCSP)			fuzzy GENET(DCSP_2)			Distance difference
		Distance	Iterations	Time (s)	Distance	Iterations	Time (s)	
150-10-10-10	90%	11.794	9212.6	0.77	11.809	7920.6	0.68	−0.015
	80%	20.043	24241.4	2.11	20.087	21552.5	1.99	−0.044
	70%	27.556	32677.1	3.13	27.567	33437.5	3.32	−0.011
	60%	33.855	35289.5	3.60	33.927	32948.8	3.52	−0.072
	50%	38.722	36244.2	3.92	38.735	33221.2	3.67	−0.013
150-10-15-10	90%	19.501	27543.6	2.82	19.476	27566.7	2.86	0.025
	80%	32.117	28579.4	3.31	32.171	29407.1	3.47	−0.054
	70%	41.879	33449.9	4.18	41.871	31940.3	4.08	0.008
	60%	49.426	38716.2	5.14	49.287	37677.1	5.11	0.139
	50%	55.855	42969.6	5.95	55.804	39129.2	5.53	0.051
150-10-10-15	90%	21.453	30997.5	3.33	21.437	29959.9	3.19	0.016
	80%	35.075	33179.0	4.05	35.076	32373.2	3.92	−0.001
	70%	44.458	38856.5	5.08	44.504	34627.2	4.50	−0.046
	60%	52.294	34339.3	4.82	52.308	36632.4	5.06	−0.014
	50%	57.604	46125.1	6.53	57.514	41792.8	6.00	0.090

Table 5.7: Comparison between fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) on randomly-generated binary DCSPs of different constraint density and tightness, using Hamming Distance

Problem	$\lvert C' \rvert : \lvert C \rvert$	fuzzy GENET(DCSP)			fuzzy GENET($DCSP_2$)			Distance difference
		Distance	Iterations	Time (s)	Distance	Iterations	Time (s)	
150-10-10-10	90%	21.129	59333.1	5.24	18.871	11481.4	4.37	2.258
	80%	38.332	62665.4	6.28	30.933	18307.0	7.00	7.399
	70%	54.423	65973.3	7.41	40.748	25883.9	9.82	13.675
	60%	67.129	65036.3	7.87	48.675	34803.4	12.82	18.454
	50%	76.068	69371.9	8.87	54.501	37111.0	14.06	21.567
150-10-15-10	90%	46.966	53623.4	6.19	41.152	26641.6	11.00	5.814
	80%	76.539	51110.8	6.76	61.653	47346.8	18.92	14.886
	70%	99.908	52228.1	7.64	77.764	56530.7	22.40	22.144
	60%	116.223	61943.0	9.55	88.590	60493.0	24.26	27.633
	50%	129.790	61293.3	9.92	97.964	62478.1	24.63	31.826
150-10-10-15	90%	52.610	54968.0	6.80	45.387	29015.9	12.35	7.223
	80%	85.300	56546.1	8.01	68.448	49877.9	21.01	16.852
	70%	107.002	56917.6	8.72	83.288	57897.8	23.76	23.714
	60%	124.273	59631.4	9.75	95.408	63069.9	26.48	28.865
	50%	136.067	62214.8	10.51	103.770	65703.4	26.64	32.297

Table 5.8: Comparison between fuzzy GENET(DCSP) and fuzzy GENET($DCSP_2$) on randomly-generated binary DCSPs of different constraint density and tightness, using Manhattan distance

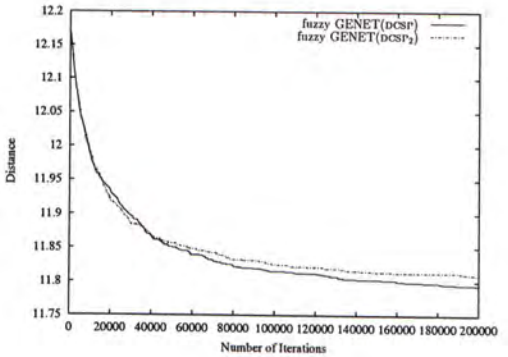
manding Manhattan distance function is used.

Progresses of distance achievement of all the problems are plotted in Figures 5.5 to 5.16. In all the problems that use Hamming distance, it is not surprising that the curves of both methods are very near to each other, since the distance achieved and effort spent are similar. Graphs on the Manhattan distance problems are more intriguing. Although the distance of fuzzy GENET(DCSP$_2$) are always lower than fuzzy GENET(DCSP), showing that the solutions of fuzzy GENET(DCSP$_2$) are more stable than those of fuzzy GENET(DCSP) all the time, its improvement in solution stability slows down suddenly at the beginning and only after a short while that it proceeds normally like in earlier experiments. It reveals that after fuzzy GENET(DCSP$_2$) has found the first few solutions, it becomes reluctant to improve until the algorithm has been run for some time. We will present an explanation in Section 5.5.

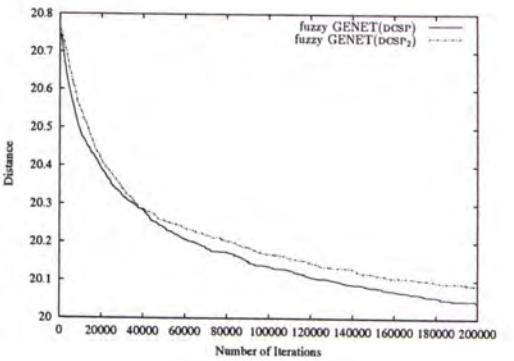
5.4.4 Variation in Domain Size

The aim of the final set of experiment is to study the algorithms' behaviours on problems with large domain size. We increase the domain size from 10 to 50 and keep the other parameters and settings identical except the threshold, which is now 0.02 $(1 - 49/50)$ according to Equation 4.5. Although the domain size is larger, the 150-50-15-15 problem is in general easier than 150-10-15-15 in the sense of CSP solving, because the expected number of solutions of 150-50-15-15 is more than that of 150-10-15-15 by about 10^{100} times, but the search space is only 5 times larger. Results on Hamming distance and Manhattan distance experiments are shown in Tables 5.9 and 5.10 respectively.

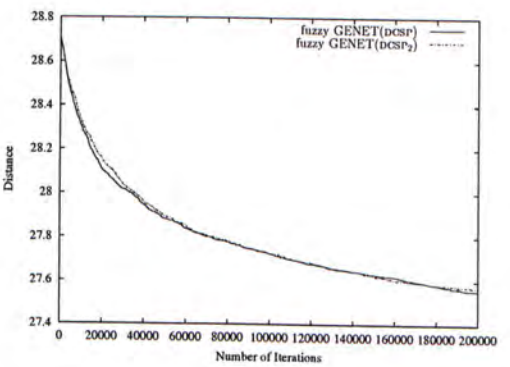
Like previous experiments on the 150-10-10-10, 150-10-15-10 and 150-10-10-15 problems, when the distance function is Hamming distance, solution stability of fuzzy GENET(DCSP) and fuzzy GENET(DCSP$_2$) are very close: the absolute difference in distance is ranging from -0.061 to 0.066. However, when Manhattan distance is used, the difference between them becomes obvious. Solutions of



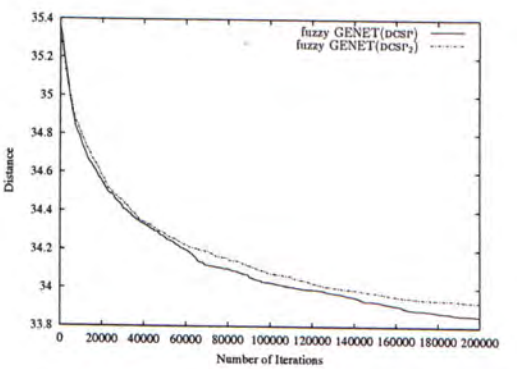
(a) 90%



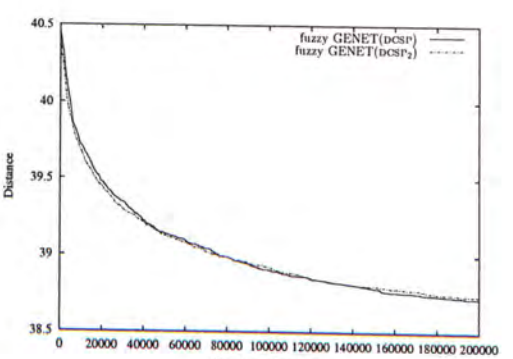
(b) 80%



(c) 70%

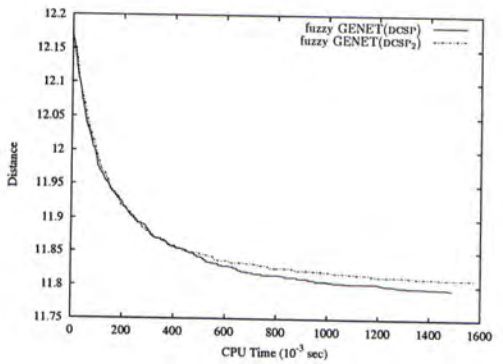


(d) 60%

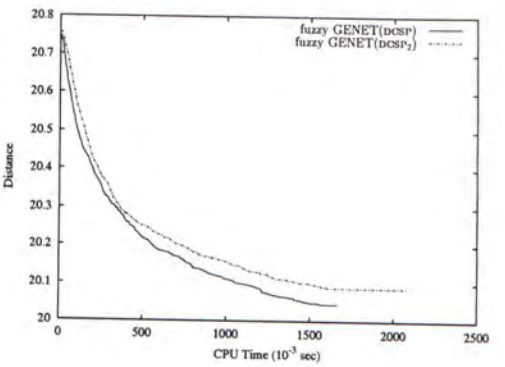


(e) 50%

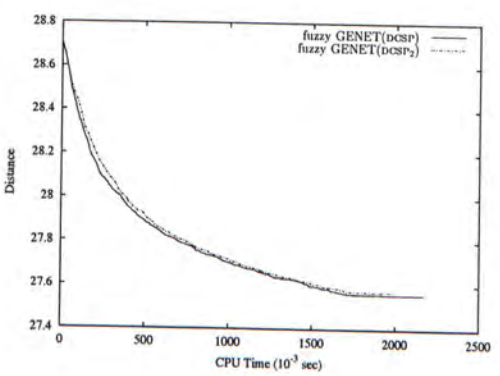
Figure 5.5: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-10-10 problem, using Hamming distance



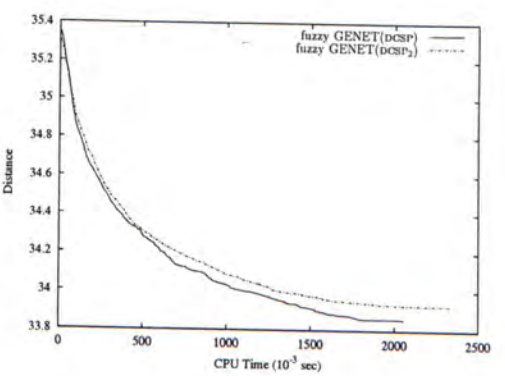
(a) 90%



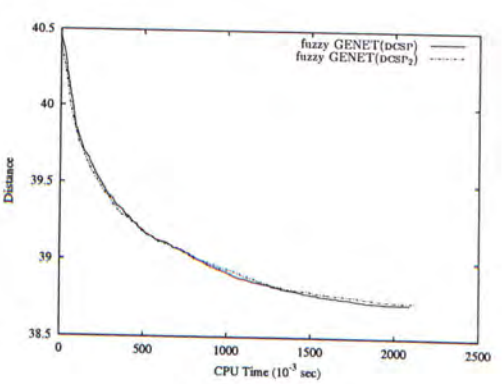
(b) 80%



(c) 70%

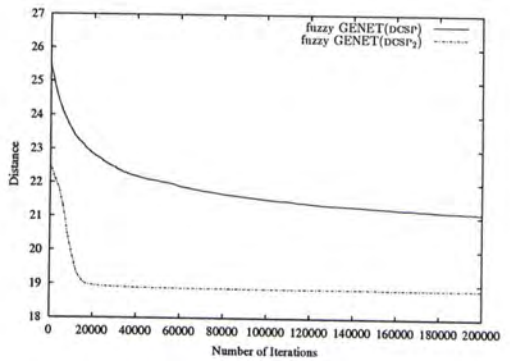


(d) 60%

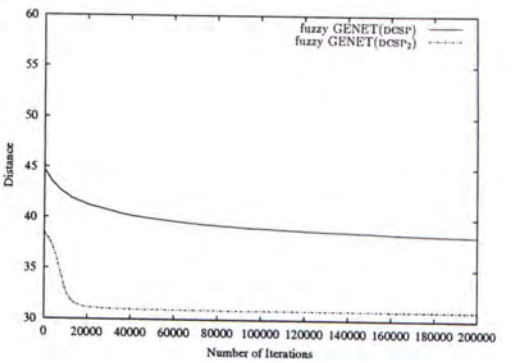


(e) 50%

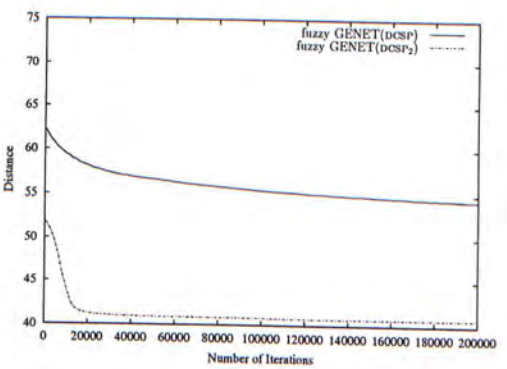
Figure 5.6: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-10-10 problem, using Hamming distance



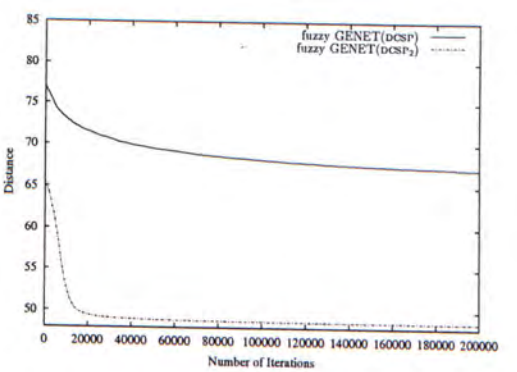
(a) 90%



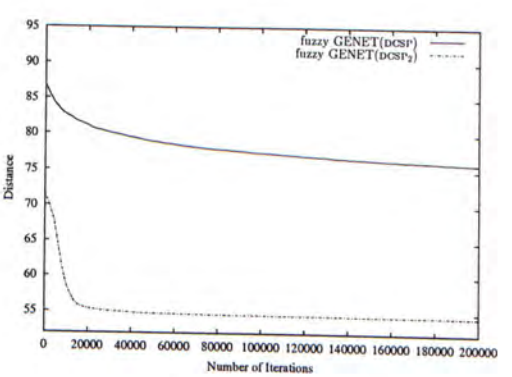
(b) 80%



(c) 70%

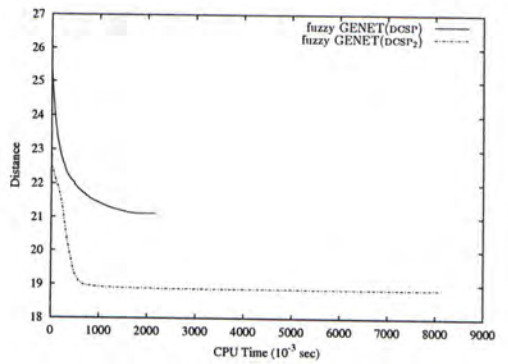


(d) 60%

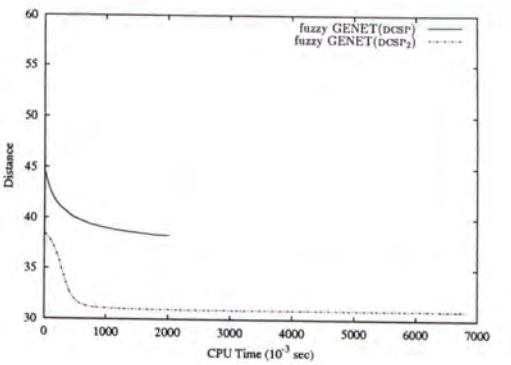


(e) 50%

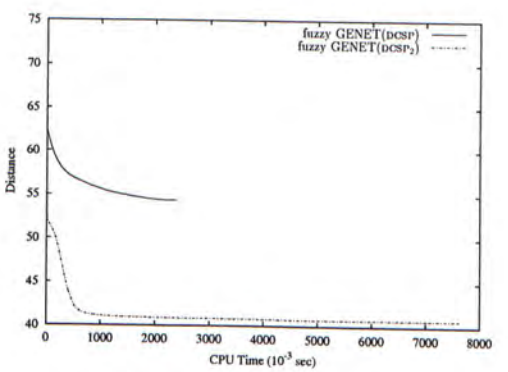
Figure 5.7: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-10-10 problem, using Manhattan distance



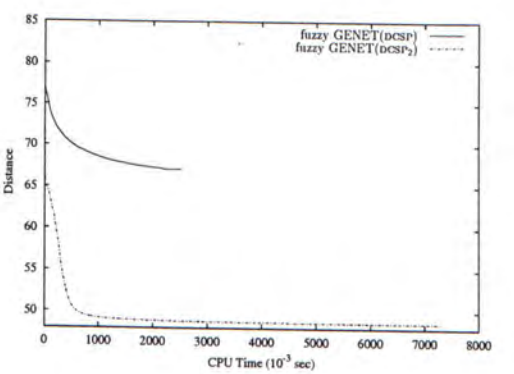
(a) 90%



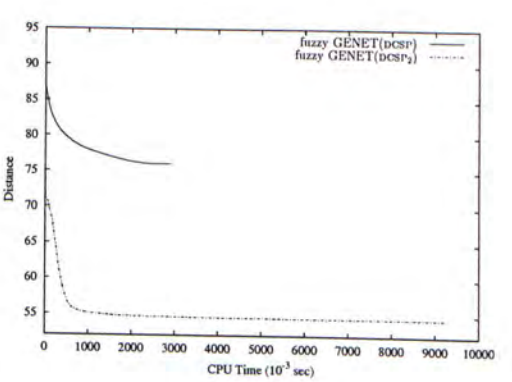
(b) 80%



(c) 70%

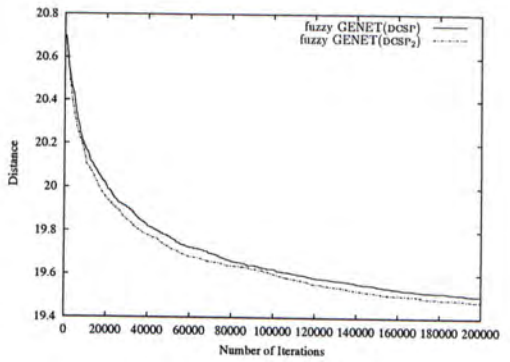


(d) 60%

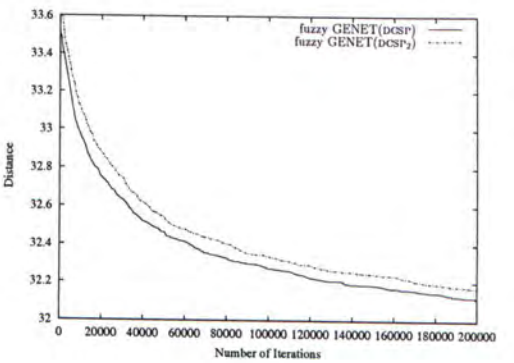


(e) 50%

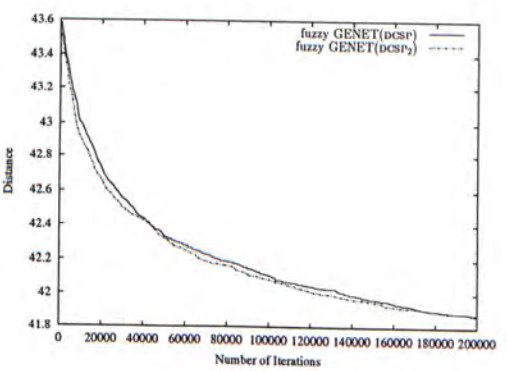
Figure 5.8: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-10-10 problem, using Manhattan distance



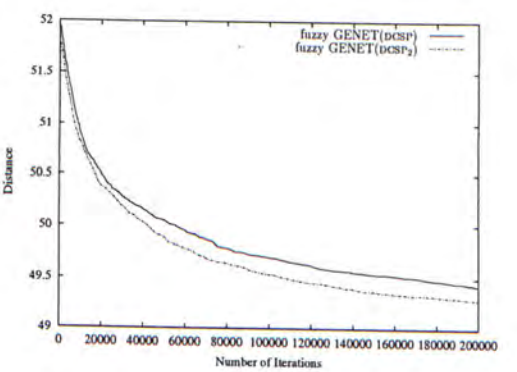
(a) 90%



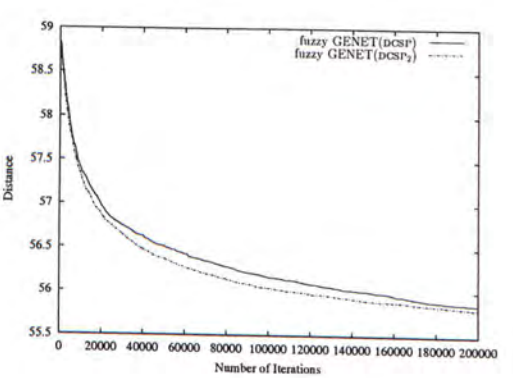
(b) 80%



(c) 70%

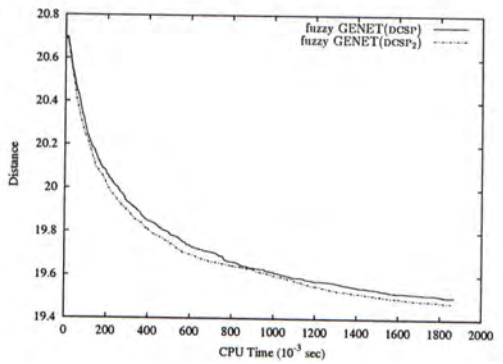


(d) 60%

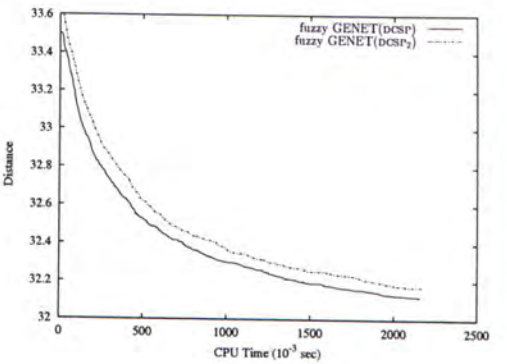


(e) 50%

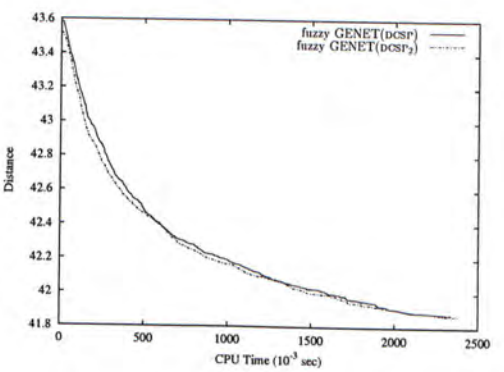
Figure 5.9: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-15-10 problem, using Hamming distance



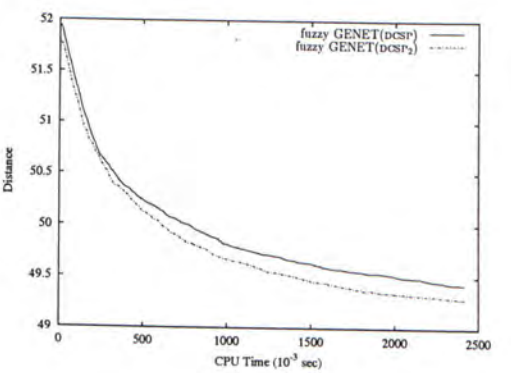
(a) 90%



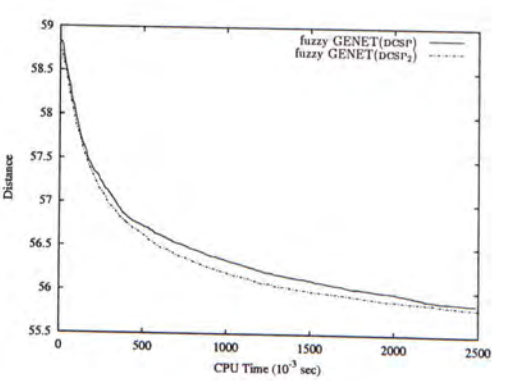
(b) 80%



(c) 70%

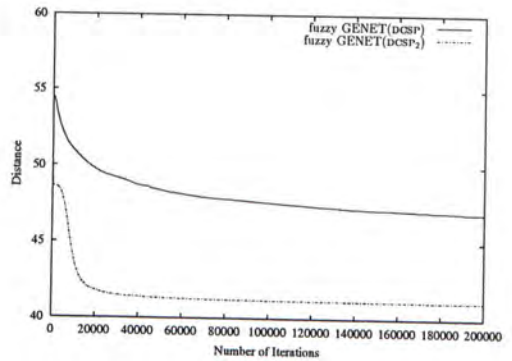


(d) 60%

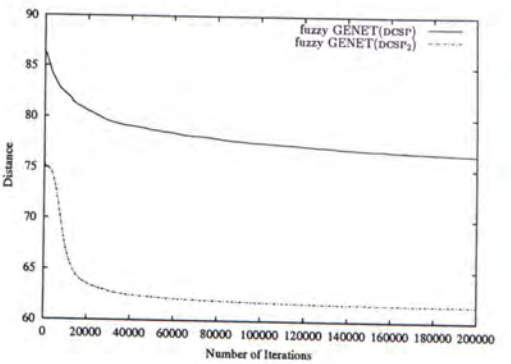


(e) 50%

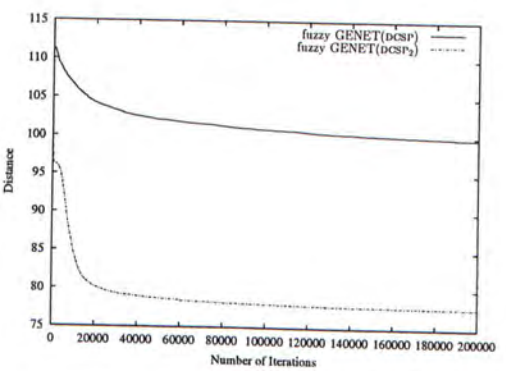
Figure 5.10: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-15-10 problem, using Hamming distance



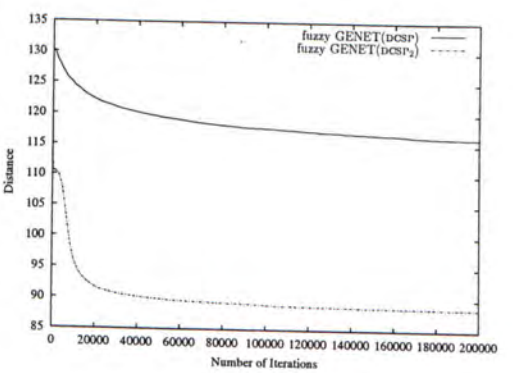
(a) 90%



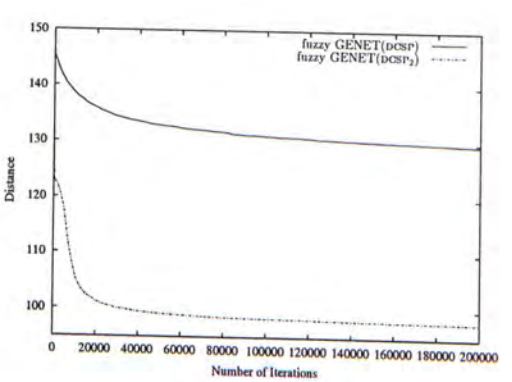
(b) 80%



(c) 70%

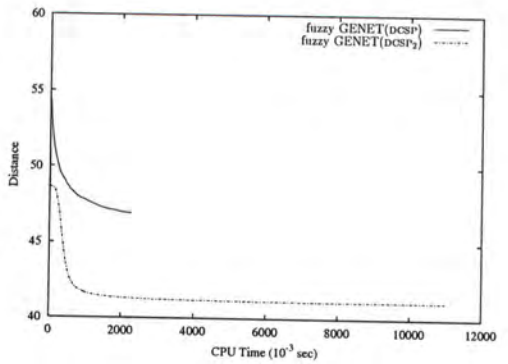


(d) 60%

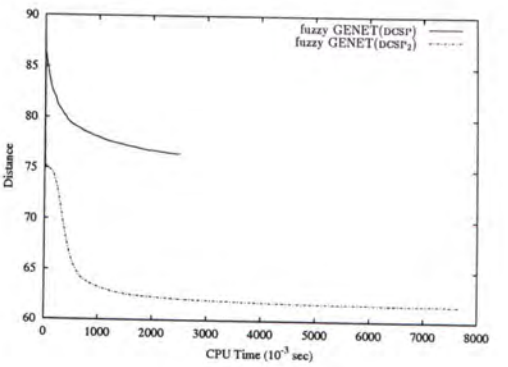


(e) 50%

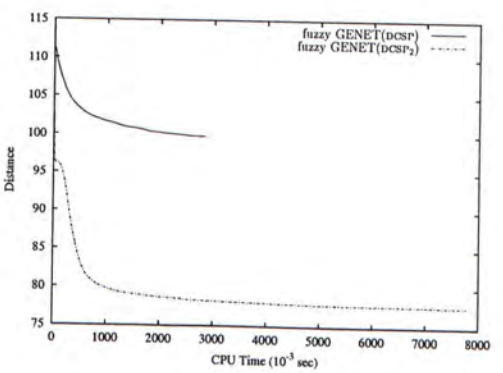
Figure 5.11: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-15-10 problem, using Manhattan distance



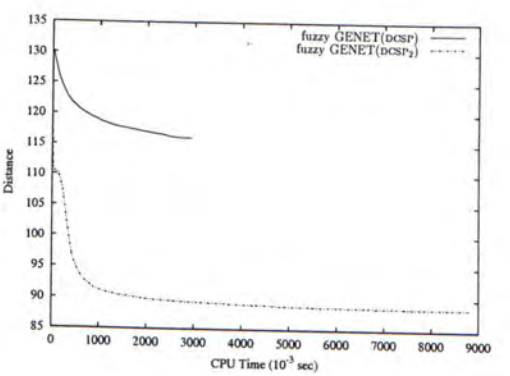
(a) 90%



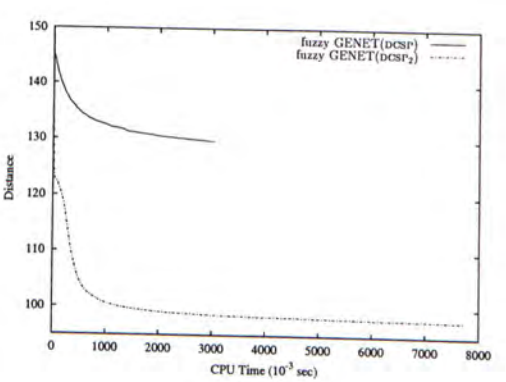
(b) 80%



(c) 70%

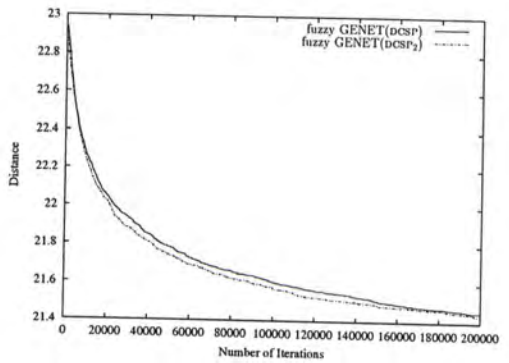


(d) 60%

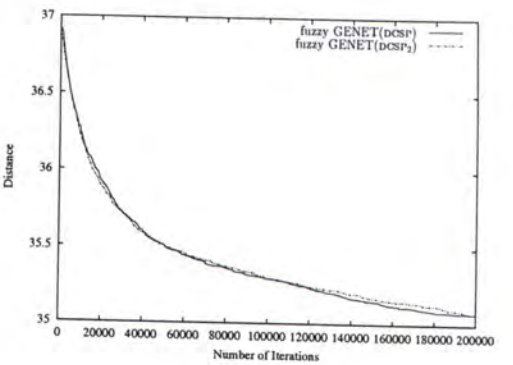


(e) 50%

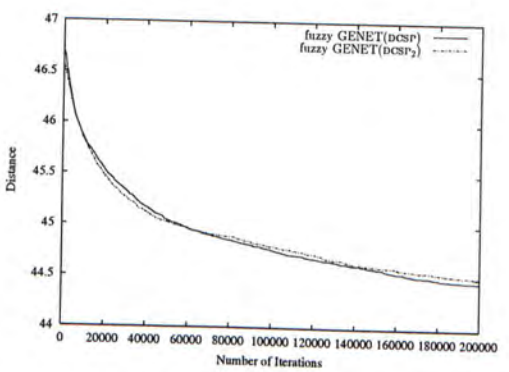
Figure 5.12: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-15-10 problem, using Manhattan distance



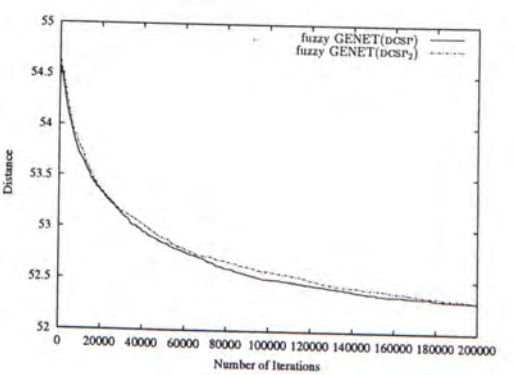
(a) 90%



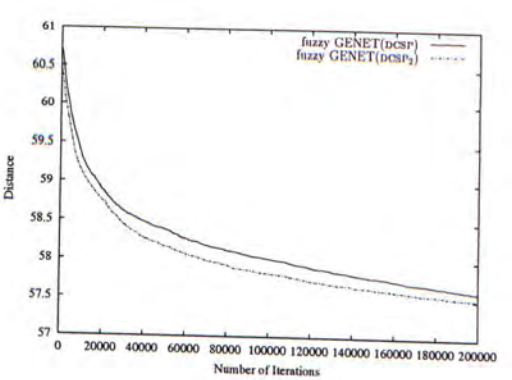
(b) 80%



(c) 70%

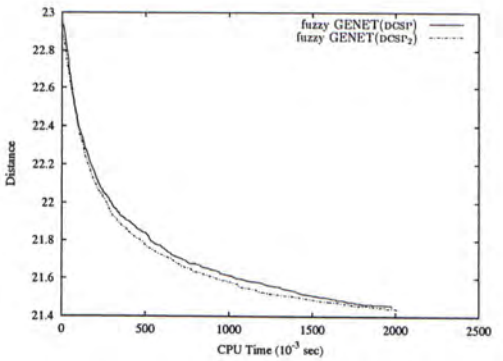


(d) 60%

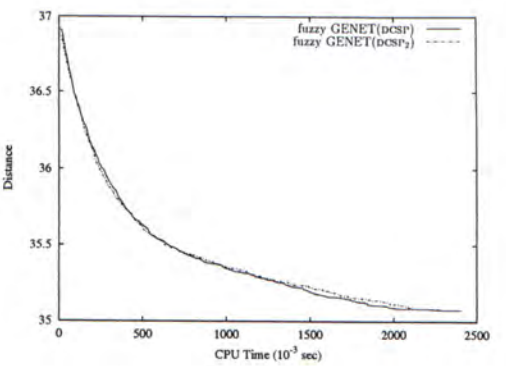


(e) 50%

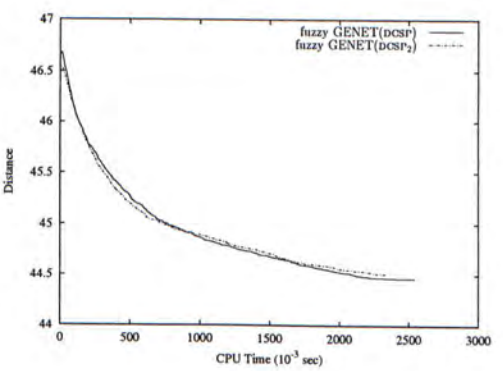
Figure 5.13: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-10-15 problem, using Hamming distance



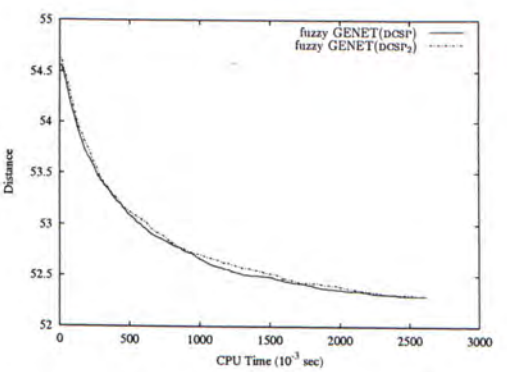
(a) 90%



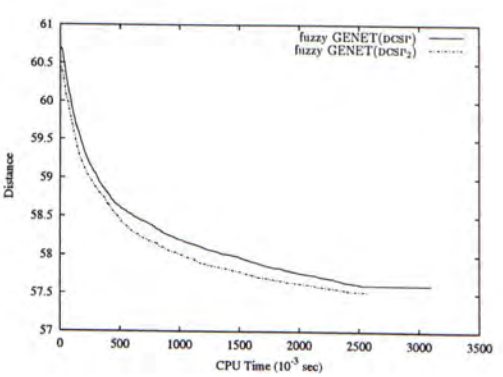
(b) 80%



(c) 70%

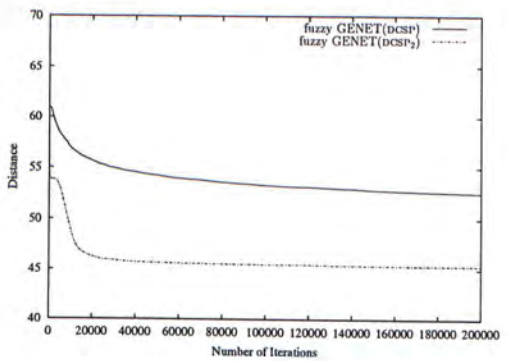


(d) 60%

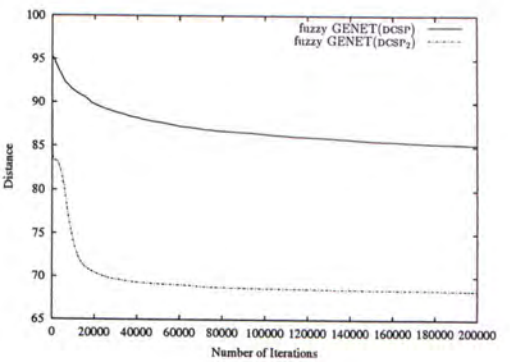


(e) 50%

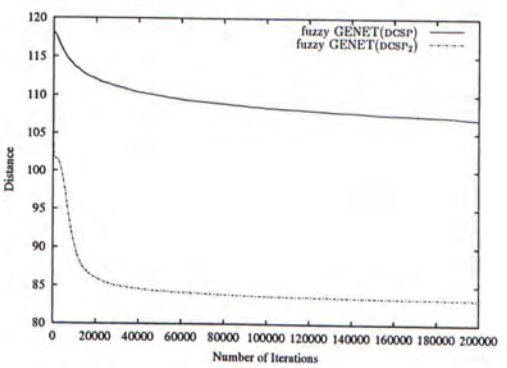
Figure 5.14: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-10-15 problem, using Hamming distance



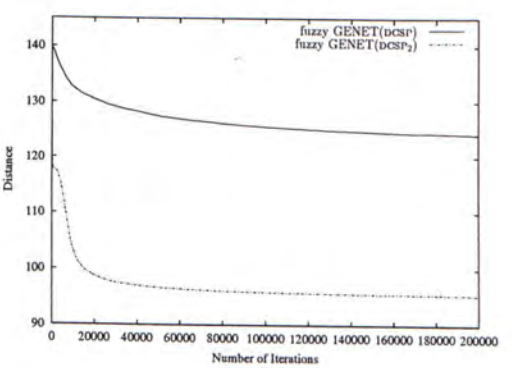
(a) 90%



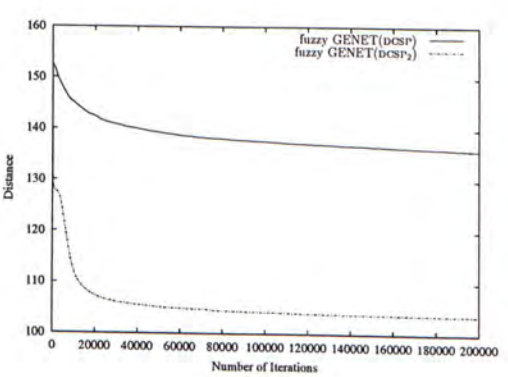
(b) 80%



(c) 70%

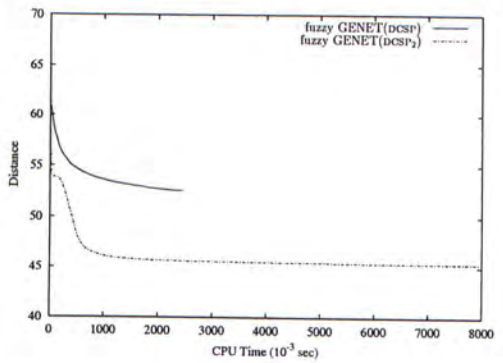


(d) 60%

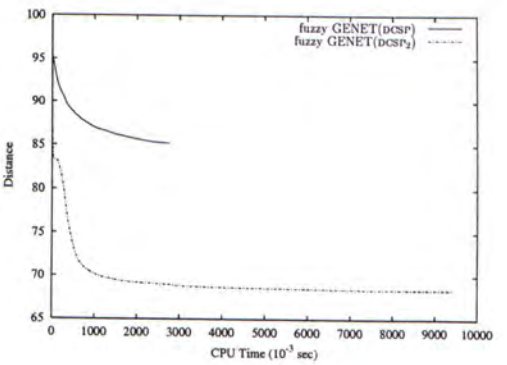


(e) 50%

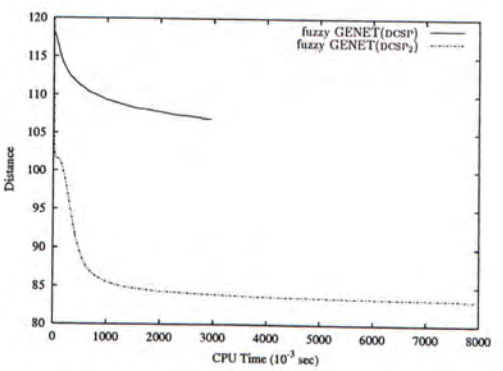
Figure 5.15: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-10-10-15 problem, using Manhattan distance



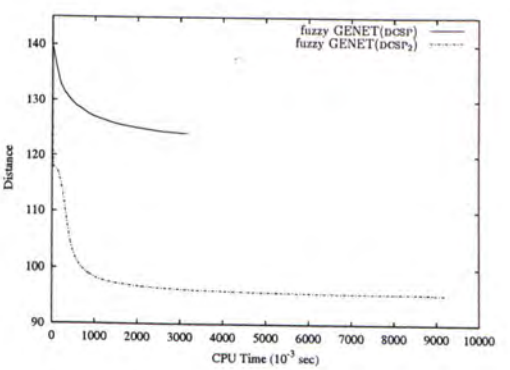
(a) 90%



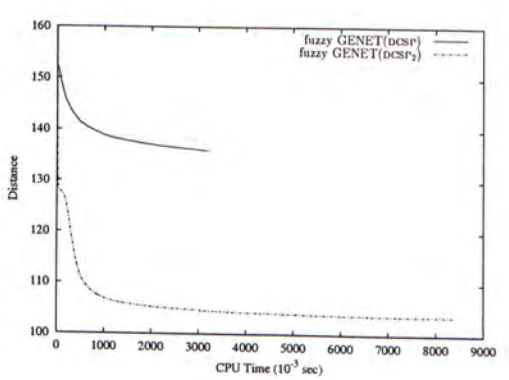
(b) 80%



(c) 70%



(d) 60%



(e) 50%

Figure 5.16: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-10-10-15 problem, using Manhattan distance

$\lvert C' \rvert / \lvert C \rvert$	fuzzy GENET(DCSP)			fuzzy GENET(DCSP_2)			Dist.
	Dist.	Iter.	Time(s)	Dist.	Iter.	Time(s)	diff.
90%	24.067	55494.4	47.73	24.047	52755.6	45.90	0.020
80%	39.620	84839.6	78.37	39.554	88312.7	81.95	0.066
70%	49.044	95290.9	98.99	48.980	92410.0	94.79	0.064
60%	56.155	97817.9	106.95	56.216	106141.0	124.82	-0.061
50%	61.208	98636.9	118.28	61.189	104924.0	115.69	0.019

Table 5.9: Comparison between fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) on the 150-50-15-15 randomly-generated binary DCSPs, using Hamming distance

$\lvert C' \rvert / \lvert C \rvert$	fuzzy GENET(DCSP)			fuzzy GENET(DCSP_2)			Dist.
	Dist.	Iter.	Time(s)	Dist.	Iter.	Time(s)	diff.
90%	146.116	75413.4	69.19	94.114	118287.0	342.53	52.002
80%	250.930	86791.8	90.40	143.986	122239.0	357.58	106.944
70%	307.872	91297.7	102.14	167.020	126274.0	370.21	140.852
60%	343.366	96090.4	110.93	180.038	128212.0	374.89	163.328
50%	370.294	99097.1	118.79	191.950	128603.0	384.36	178.344

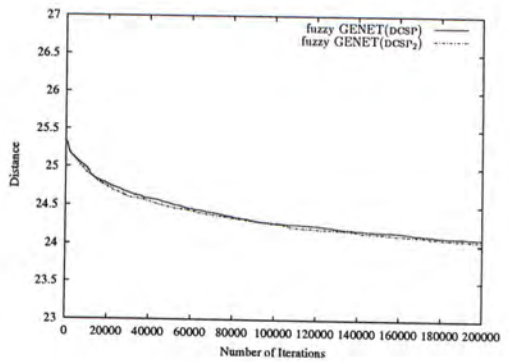
Table 5.10: Comparison between fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) on the 150-50-15-15 randomly-generated binary DCSPs, using Manhattan distance

fuzzy GENET(DCSP_2) are much more stable, the absolute difference in distance is ranging from about 52.002 to 178.344.

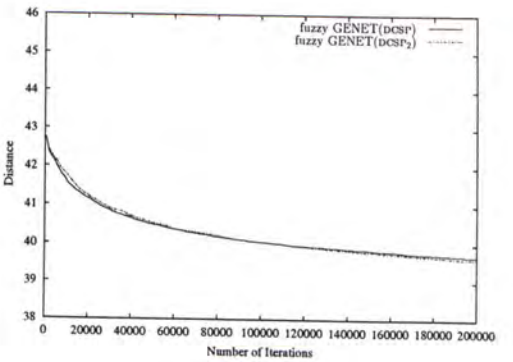
In order to study their performance, progresses of distance achieved in this set of experiments are plotted in Figures 5.17, 5.18, 5.19 and 5.20.

For the Hamming distance experiments, both methods again progress "hand in hand" during the whole program execution. In the Manhattan distance experiments, fuzzy GENET(DCSP_2) obtains more stable results than fuzzy GENET(DCSP) from the start and keep it until the programs finish. We also notice an irregularity in the curves of fuzzy GENET(DCSP_2) that they stay "flat" at the beginning for a longer time than in the previous 150-variables experiments. Section 5.5 will try to answer this question.

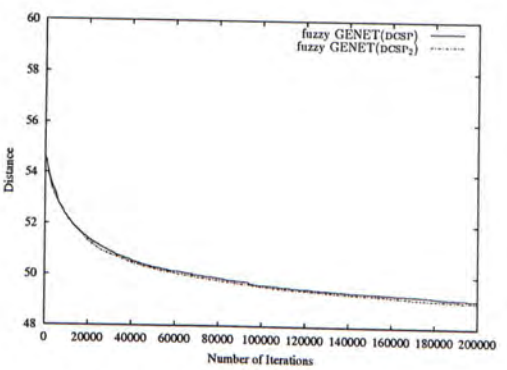
We conclude that for a problem having moderately large domain size, both methods perform equally well in terms of solution quality when a simple distance function like Hamming distance is used. When a more demanding distance



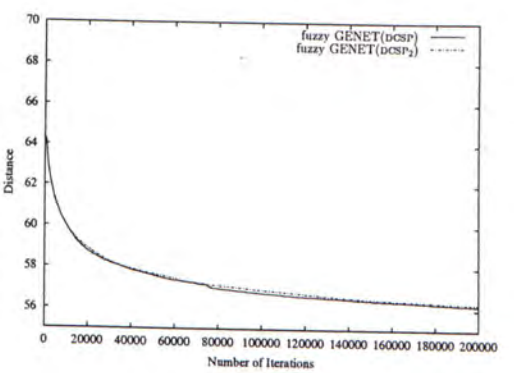
(a) 90%



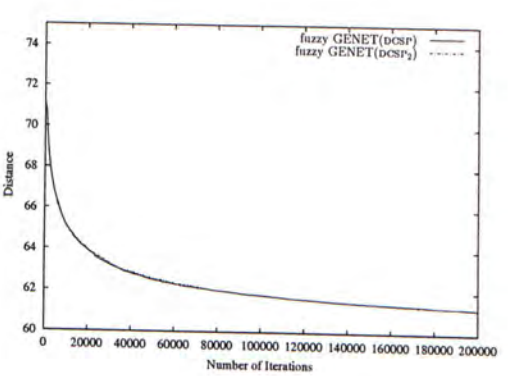
(b) 80%



(c) 70%

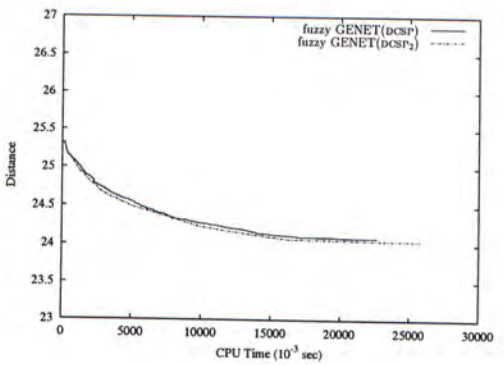


(d) 60%

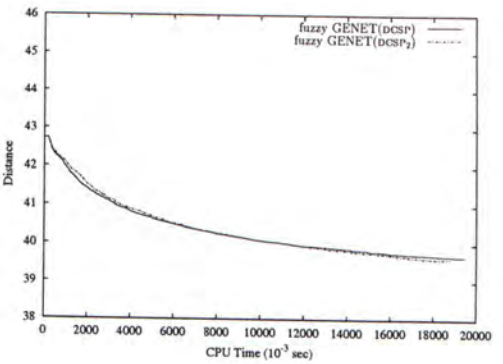


(e) 50%

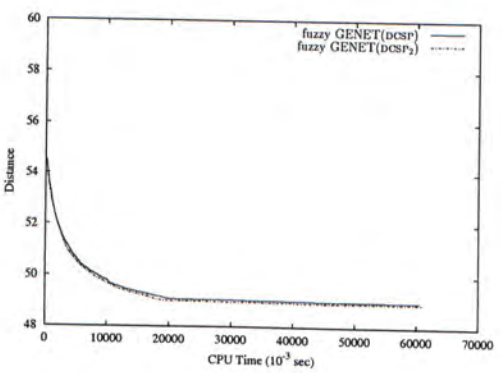
Figure 5.17: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-50-15-15 problem, using Hamming distance



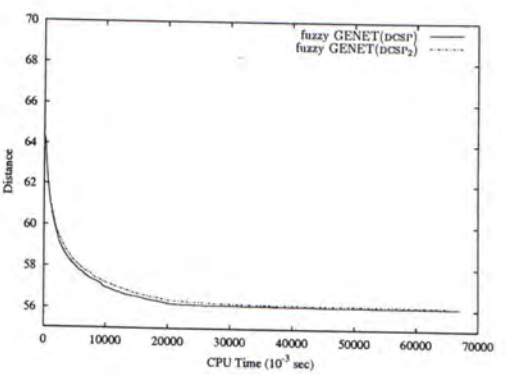
(a) 90%



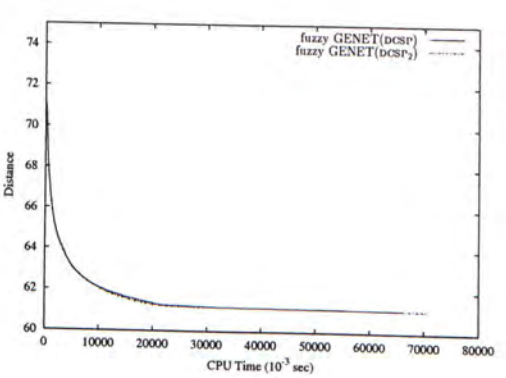
(b) 80%



(c) 70%

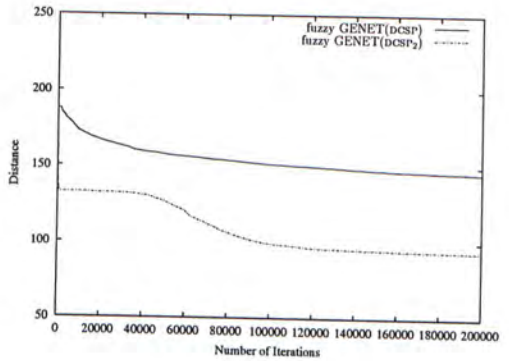


(d) 60%

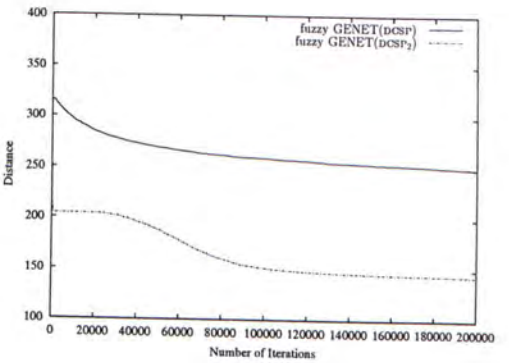


(e) 50%

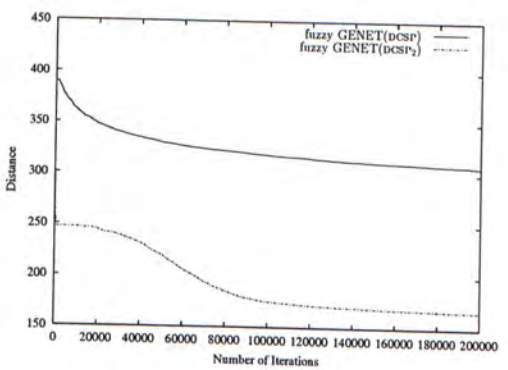
Figure 5.18: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-50-15-15 problem, using Hamming distance



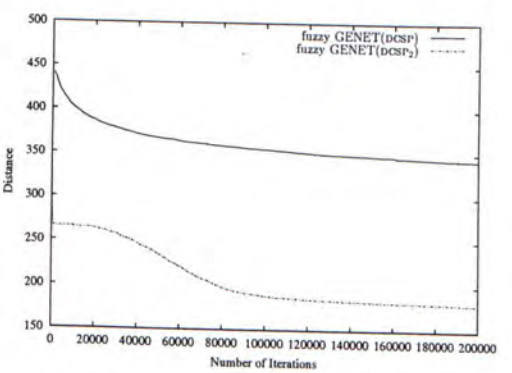
(a) 90%



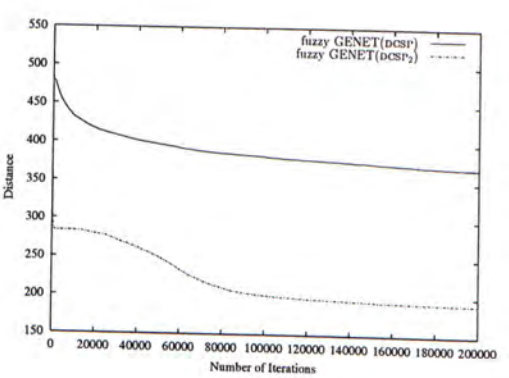
(b) 80%



(c) 70%

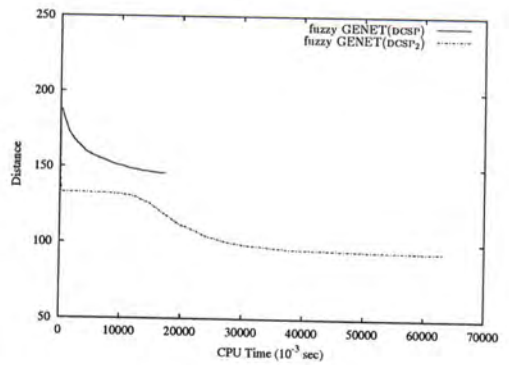


(d) 60%

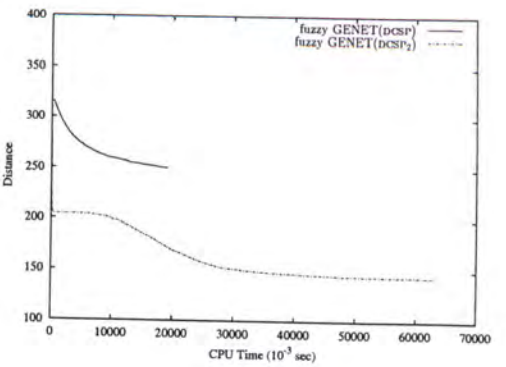


(e) 50%

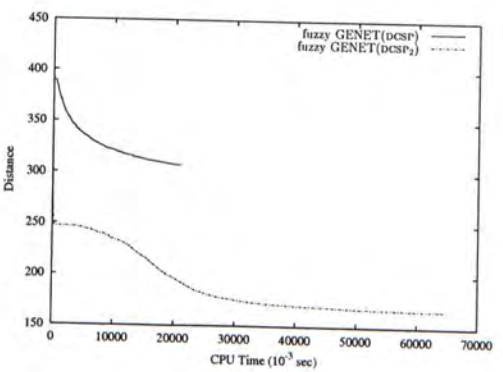
Figure 5.19: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the number of iterations in the 150-50-15-15 problem, using Manhattan distance



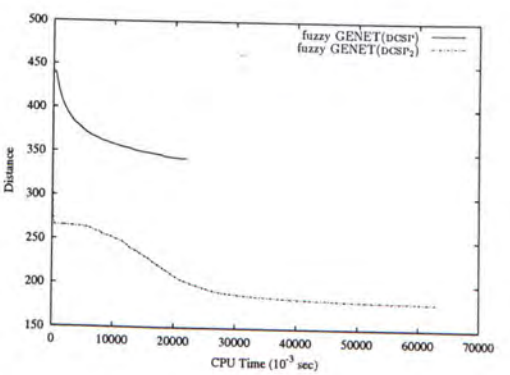
(a) 90%



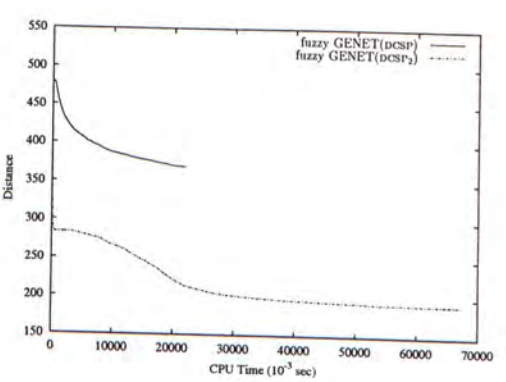
(b) 80%



(c) 70%



(d) 60%



(e) 50%

Figure 5.20: Comparison of distance achieved by fuzzy GENET(DCSP) and fuzzy GENET(DCSP_2) against the CPU time taken in the 150-50-15-15 problem, using Manhattan distance

function, such as the Manhattan distance, is used, solution quality of fuzzy GENET(DCSP$_2$) can surpass that of fuzzy GENET(DCSP) by a large degree.

5.5 Analysis of Fuzzy GENET(DCSP$_2$)

In some of the experiments, fuzzy GENET(DCSP$_2$) exhibits a slowdown in distance improvement during the early moment of program execution, but after a while it gradually gains momentum. It is noticeable in the experiments on the 150-10-10-10, 150-10-10-15 and 150-10-15-10 problems using the Manhattan distance, and most prominently in the 150-50-15-15 problem.

We conjecture that this is because the distance bound is too "tight" to the search such that its movement is too restricted. The purpose of the distance bound is to establish a barrier on the landscape to confine the search to be within the areas that contain better solutions. However if the area being searched does not have a better solution, the search will be forced to enter a local maximum. Heuristic learning will then be applied and it will eventually come out of the confined area and continue in other places.

If the search is too confined in an area by the distance bound, it will spend a lot of time inside it before coming out from it by the help of the heuristic learning rule. On the other hand, if the "height" of this barrier can be lowered, we believe that this wasted heuristic learning cycle can be avoided since the search will become easier to retract to variable assignments of *poorer* stability but with fewer constraint violations.

Consider the following example: we are solving a stability problem, of which the CSP has 150 variables and a uniform domain size of 10, and uses Manhattan distance. The best distance found so far is 301. The algorithm has been run for some time already, and the weights of the constraints are as shown in Table 5.11. We are now selecting a value for variable z, and it turns out that 3 is chosen because the input $I_{\langle z,3\rangle}$ is the highest.

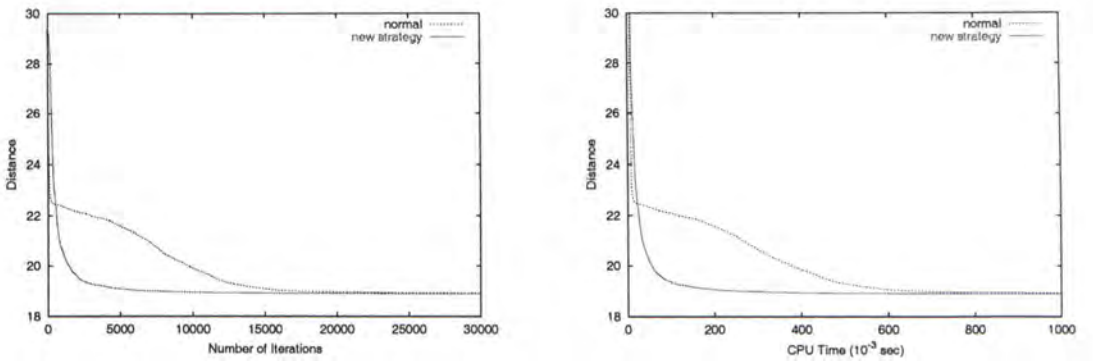
Value	Distance	Degree of satisfaction	Weight		Input
			distance bound	others	
0	304	0.77800	−0.22200	−6.0	−6.22200
1	303	0.77726	−0.22274	−7.0	−7.22274
2	302	0.77652	−0.22348	−7.0	−7.22348
3	301	0.77578	−0.22422	−5.0	**−5.22422**
4	300	0.77504	−0.22496	−5.0	−5.22496
5	301	0.0	−1.0	−5.0	−6.0
6	302	0.0	−2.0	−4.0	−6.0
7	303	0.0	−3.0	−4.0	−7.0
8	304	0.0	−3.0	−4.0	−7.0
9	305	0.0	−2.0	−6.0	−8.0

Table 5.11: Weights of variable z in our hypothetic example under fuzzy GENET(DCSP$_2$)

Value	Distance	Degree of satisfaction	Weight		Input
			distance bound	others	
0	304	0.77800	−0.22200	−6.0	−6.22200
1	303	0.77726	−0.22274	−7.0	−7.22274
2	302	0.77652	−0.22348	−7.0	−7.22348
3	301	0.77578	−0.22422	−5.0	−5.22422
4	300	0.77504	−0.22496	−5.0	−5.22496
5	301	0.0	−0.22570	−5.0	−5.22570
6	302	0.0	−0.45288	−4.0	**−4.45288**
7	303	0.0	−0.68154	−4.0	−4.68154
8	304	0.0	−0.68376	−4.0	−4.68376
9	305	0.0	−0.45732	−6.0	−6.45732

Table 5.12: Weights of variable z in our hypothetic example if weights of tuples in distance bound are not reset to -1 and tuples are penalized by their original weights

If, however, weights of all tuples in the distance bound are retained when a better solution is found, in other words no tuples will have their weights re-initialized to -1, and when penalizing them the original values of their weights are used instead of using -1, we will have the values in Table 5.12. In this case, domain value 6 is chosen instead of 3. The purpose of this strategy is to lower the barrier of the distance bound, because the weights are now less negative. As expected, values that lead to poorer stability (but with few constraint violations) will now have a higher chance to be selected.

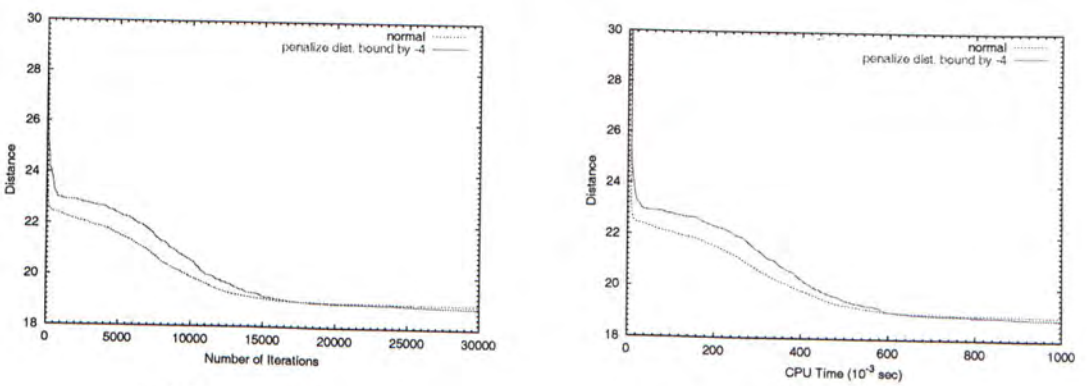


(a) by iteration (b) by time

Figure 5.21: Execution of fuzzy GENET(DCSP_2) on the 150-10-10-10 problem ($|C'| : |C| = 90\%$, Manhattan distance) when weights of tuples in distance bound are not reset to -1 and tuples are penalized by their original weights

As a proof of concept, we employ the strategy just mentioned to test the 150-10-10-10 problem ($|C'| : |C| = 90\%$) using Manhattan distance again. Results are plotted in Figure 5.21. Interestingly enough, solution quality is improved in a much more rapid rate, and the best solution is just as good as that found by the original scheme.

To complete our study, we tested the opposite of this strategy by enlarging the weights, which are reset to -4 instead of -1, and are penalized by adding -4 each time. Figure 5.22 shows that it really does worsen the performance.



(a) by iteration

(b) by time

Figure 5.22: Execution of fuzzy GENET(DCSP$_2$) on the 150-10-10-10 problem ($|C'| : |C| = 90\%$, Manhattan distance) when tuples in distance bound are reset to and penalized by -4

Chapter 6

Conclusion

We end the thesis by summarizing our contributions in this research and presenting possible directions for future work.

6.1 Contributions

Our major contribution is we propose a fuzzy constraint satisfaction approach to solve stability problems effectively and efficiently.

This dissertation describes two schemes to model a stability problem as a fuzzy CSP. The first one is to impose supplementary unary fuzzy constraints into a modified CSP. Users can express their criteria of a stable solution with the help of the degrees of satisfaction in the tuples of these supplementary constraints. Fuzzy GENET is modified to make it successively look for better solutions until stopping criteria are met. In this regard, fuzzy GENET(DCSP) is an anytime search algorithm [9, 51], for the stability of the solutions can be improved as long as more time is given. Experimental results on small problems show that fuzzy GENET(DCSP) is a good anytime search algorithm, as initial solutions can be found quickly and are of reasonable quality. The results also illustrate that the final results are very close to the optimal solutions.

As the first approach cannot model any distance metrics, we then improve

the first modelling formulation by replacing the unary fuzzy constraints with an n-ary fuzzy constraint called distance bound. The distance bound constraint makes modelling stability problems simpler and more flexible. Fuzzy GENET(DCSP) is further enhanced by incorporating a strategy to limit the search to only look for better solutions, which is coincidentally similar to the idea of iterative deepening [17]. We show that this approach achieves solutions that are at least as stable as those found by the first one, and are substantially better in some cases. The second approach is also more efficient than the first one most of the time as seen from the convergence graphs. Its advantage is more obvious when the problem size is large and when the Manhattan distance is used.

6.2 Future Work

As pointed out in Section 5.5, fuzzy GENET(DCSP_2) does not converge well for the large-size problems at the beginning of program execution, especially in the 150-50-15-15 case. Explanations and preliminary testing have been presented to show that retaining the weights of the tuples in the distance bound and decreasing the value for penalization may improve the effectiveness and efficiency of the algorithm. An extensive experimentation and more in-depth analysis are needed to ensure the usefulness of this approach. Moreover, the reason of the irregularities shown in Figure 4.3(c) is still unknown to us, and investigation should be carried out.

Since carrying out this research is motivated by the practical necessity in real-life problems, it is worthwhile to test our proposals on real-life applications, such as the train rescheduling problem [7] and the ones mentioned in Section 2.2. However, fuzzy GENET(DCSP_2) is not powerful enough to solve many kinds of these applications because it can only handle binary constraints. Decomposing non-binary constraints into binary ones is impractical due to the possible

tremendous increase in the number of constraints. Hence we have to first extend fuzzy GENET(DCSP$_2$) to handle non-binary constraints. E-GENET [48, 18] is an enhancement of GENET to remedy this shortcoming of GENET. Many of its ideas can be applied to fuzzy GENET(DCSP$_2$), most notably the concept of constraint node and intermediate node. The ability to efficiently handle the linear arithmetic constraints[1] and general constraints, for instance the `illegal` constraint, the `atmost` constraint and the `among` constraint, is also very desirable, because they often appear in real-world applications.

It is also interesting to discover some applications with unique stability requirements to examine how the algorithm performs on a wider range of distance metrics.

[1] A *linear arithmetic constraint* is of the form $X \circ Y$, where X and Y are linear arithmetic expressions and $\circ \in \{=, \neq, <, \leq, >, \geq\}$. X and Y can be written as $A_0 + A_1 u_1 + \cdots + A_k u_k$, where each $A_i, 0 \leq i \leq k$ is an integer and $u_j, 1 \leq j \leq k$ is a variable.

Bibliography

[1] A. Bellicha. Maintenance of solution in a dynamic constraint satisfaction problem. In G. Rzevski, J. Pastor, and R. Adey, editors, *Applications of Artificial Intelligence in Engineering VIII*, Southampton and London, 1993. Computational Mechanics Publications with Elsevier Applied Science.

[2] Christian Bessière. Arc-consistency in dynamic constraint satisfaction problems. In Kathleen Dean, Thomas L.; McKeown, editor, *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 221–226. MIT Press, July 1991.

[3] Christian Bessière. Arc-consistency for non-binary dynamic CSPs. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 23–27, Vienna, Austria, August 1992. John Wiley & Sons.

[4] Chritian Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1):179–190, January 1994.

[5] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Integrated planning and execution for autonomous spacecraft. In *Proceedings of the IEEE Aerospace Conference (IAC)*, Aspen CO, March 1999.

[6] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to increase the responsiveness of planning and scheduling for autonomous spacecraft. In *IJCAI99 Workshop on Scheduling and Planning*

meet Real-time Monitoring in a Dynamic and Uncertain World, Stockholm, Sweden, August 1999.

[7] C. K. Chiu, C. M. Chou, J. H. M. Lee, H. F. Leung, and Y. W. Leung. A constraint-based interactive train rescheduling tool. In *Second International Conference on Principles and Practice of Constraint Programming*, pages 104–118, August 1996.

[8] Andrew Davenport, Edward Tsang, Chang J. Wang, and Kangmin Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of the 12th National Conference on Artificial Intelligence. Volume 1*, pages 325–330, Menlo Park, CA, USA, July31 August–4 1994. AAAI Press.

[9] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 49–54, Saint Paul, Minnesota, USA, 1988. AAAI Press/MIT Press.

[10] Romuald Debruyne. Arc-consistency in dynamic CSPs is no more prohibitive. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 299–307. IEEE Computer Society Press, November 1996.

[11] Rina Dechter and Avi Dechter. Belief maintenance in dynamic constraint networks. In Tom M. Smith, Reid G.; Mitchell, editor, *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 37–42, St. Paul, MN, August 1988. Morgan Kaufmann.

[12] T. Estlin, G. Rabideau, D. Mutz, and S. Chien. Using continuous planning techniques to coordinate multiple rovers. In *IJCAI99 Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World*, Stockholm, Sweden, August 1999.

[13] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, January 1992.

[14] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, August 1993.

[15] ILOG. *ILOG Solver 4.0 Reference Manual.* ILOG, May 1997.

[16] ILOG. *ILOG Solver 4.0 User's Manual.* ILOG, May 1997.

[17] R. Korf. Heuristics as invariants and its application to learning. *Machine Learning*, pages 100–103, 1985.

[18] J.H.M. Lee, H.F. Leung, and H.W. Won. Extending GENET for non-binary CSP's. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 338–343, November 1995.

[19] J.H.M. Lee, H.F. Leung, and H.W. Won. Towards a more efficient stochastic constraint solver. In *In Second International Conference on Principles and Practice of Constraint Programming*, pages 338–352, August 1996.

[20] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[21] Pedro Meseguer and Javier Larrosa. Solving fuzzy constraint satisfaction problems. In *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems*, pages 1233–1238, July 1997.

[22] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 17–24, 1990.

[23] Steven Minton, Mark D. Johnston, Andrew B. Phillips, and Phillip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[24] Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[25] Ugo Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Sciences*, 7(2):95–132, 1974.

[26] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 40–45, Washington, DC, 1993.

[27] B. Neveu and P. Berlandier. Maintaining arc consistency through constraint retraction. In *Proceedings of 6th IEEE International Conference on Tools with Artificial Intelligence*, pages 426–431, November 6–9 1994.

[28] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, and A. Govindjee. Iterative repair planning for spacecraft operations in the aspen system. In *International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS)*, Noordwijk, The Netherlands, June 1999.

[29] Zsofi Ruttkay. Fuzzy constraint satisfaction. In *Proceedings of the Third IEEE International Conference on Fuzzy Systems*, volume 2, pages 1263–1268, June 1994.

[30] Hani El Sakkout, Thomas Richards, and Mark Wallace. Minimal perturbation in dynamic scheduling. In *European Conference on Artificial Intelligence*, pages 504–508, 1998.

[31] Hani El Sakkout, Thomas Richards, and Mark G. Wallace. Unimodular probing for minimal perturbance in dynamic resource feasibility problems. In *CP97 Workshop on the Theory and Practice of Dynamic Constraint*

Satisfaction, Salzburg, Austria, November 1997. Held in conjunction with Third International Conference on Principles and Practice of Constraint Programming (CP97).

[32] Hani El Sakkout and Mark Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

[33] Thomas Schiex. Possibilistic constraint satisfaction problems. In *Proceedings of the Eighth Conference of Uncertainty in Artificial Intelligence*, pages 269–275, Stanford, 1992.

[34] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, San Jose, California, July 1992. AAAI Press.

[35] Barbara M. Smith. The phrase transition and the mushy region in constraint satisfaction problems. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 100–104, 1994.

[36] Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction. *Artificial Intelligence*, 81(1–2):155–181, 1996.

[37] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, 1993.

[38] Gérard Verfaillie and Thomas Schiex. Dynamic backtracking for dynamic constraint satisfaction problems. In *Proceedings of the ECAI'94 Workshop on Constraint Satisfaction Issues Raised by Practical Applications*, pages 1–8, Amsterdam, The Netherlands, August 1994.

[39] Gérard Verfaillie and Thomas Schiex. Solution and reasoning reuse in space planning and scheduling applications. In *Proceedings of the Third Interna-*

tional Symposium on Artificial Intelligence, Robotics, and Automation for Space, Pasadena, California, 1994.

[40] Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceedings of the 12th National Conference on Artificial Intelligence. Volume 1*, pages 307–312, Menlo Park, CA, USA, 1994. AAAI Press.

[41] Chris Voudouris. *Guided Local Search for Combinatorial Optimisation Problems.* PhD thesis, Department of Computer Science, University of Essex, UK, July 1997.

[42] Chris Voudouris and Edward Tsang. Partial constraint satisfaction problems and Guided Local Search. In *Proceedings of Second International Conference on Practical Application of Constraint Technology*, pages 337–356, London, April 1996.

[43] Chris Voudouris and Edward Tsang. Solving the radio link frequency assignment problem using Guided Local Search. In *Proceedings of NATO Symposium on Radio Length Frequency Assignment, Sharing and Conservation Systems (Aerospace)*, Aalborg, Denmark, October 1998.

[44] Chris Voudouris and Edward Tsang. Guided local search. *European Journal of Operational Research*, 113(2):469–499, March 1999.

[45] Benjamin W. Wah and Yi Xin Chen. Optimal anytime constrained simulated annealing for constrained global optimization. In *Sixth International Conference on Principles and Practice of Constraint Programming*, pages 425–440, 2000.

[46] Benjamin W. Wah and Tao Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. In *Principles and Practice of Constraint Programming*, pages 461–475, October 1999.

[47] Richard J. Wallace and Eugene C. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *Fourth International Conference on Principles and Practice of Constraint Programming (CP98)*, pages 447–461, Pisa, Italy, 1998.

[48] Stephen Hon Wing Won. Constraint satisfaction problem solving with E-GENET. Master's thesis, Department of Computer Science and Engineering, The Chinese University of Hong Kong, 1997.

[49] Jason H. Y. Wong and H. F. Leung. Extending GENET to solve fuzzy constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 98)*, pages 380–385, Madison, Wisconsin, July 1998. AAAI Press.

[50] Jason H. Y. Wong and H. F. Leung. Solving fuzzy constraint satisfaction problems with fuzzy GENET. In *Proceedings of the 10th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'98)*, pages 180–191, Taipei, Taiwan, ROC, November 1998. IEEE Press.

[51] Shlomo Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms.* PhD thesis, Computer Science Division, University of California at Berkeley, 1993.