Constraint Satisfaction Networks: from Operational Research to Cortical Microcircuits

eingereichte
ARBEIT im HAUPTSEMINAR von

B. Eng. Johanna Seegers

geb. am 18.01.1992
wohnhaft in:
Clemensstrasse 88
80796 München
Tel.: 089 20008501

Lehrstuhl für
STEUERUNGS- und REGELUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss

Betreuer: M. Sc. Cristian Axenie
Beginn: 09.04.2014
Abgabe: 27.06.2014

Abstract

Constraint Satisfaction Problems are ubiquitous in technical problems and natural systems. Challenging tasks in scheduling, resource allocation and robotics are usually formalized as CSPs. Although existing algorithms for solving CSPs are mature and efficient for specific problems they still need exponential time for finding a solution or even finding one. Recently, neural networks receive more attention in various fields of application and models become complex and powerful. Research has shown that neural networks in human brains are capable to come up with solutions for Constraint Satisfaction Problems in an adaptive and robust manner, as human performance shows. Therefore, the implementation of neural networks as problem solvers is of high interest.
The report starts by introducing the basics of Constraint Satisfaction Problems. Then it introduces some neural implementations comparing them with classical (state-of-the-art) approaches. A more detailed analysis of neural substrate is then emphasized with focus on the main formalism of CSPs. Finally, some robotics applications scenarios are discussed and accompanied by a case study.

Zusammenfassung

Constraint Satisfaction Problems sind allgegenwärtig in technischen Problem und natürlichen Systemen. Herausfordernde Aufgabe wie Zeitplanung, Betriebsmittelzuweisung und die Robotertechnik werden für gewöhnlich als CSPs formuliert. Obwohl die existierenden Algorithmen zur Lösung von CSPs ausgereift und effizient für spezielle Probleme sind brauchen sie exponentiell lang, um zu einer Lösung zu kommen. In der letzten Zeit genießen neuronale Netze immer mehr Aufmerksamkeit in verschiedenen Anwendungsgebieten und die Modelle werden komplexer und leistungsfähiger. Forschungsergebnisse zeigen, dass neuronale Netze im menschlichen Gehirn fähig sind auf anpassungsfähige und robuste Weise CSPs zu lösen. Das zeigt sich auch in der menschlichen Leistung. Daher ist der Einsatz von neuronalen Netzen als Problemlöser von großem Interesse.
Diese Arbeit beginnt mit einer Einführung in die Grundlagen von Constraint Satisfaction Problems. Dann stellt sie einige neuronale Umsetzungen den klassischen Herangehensweisen gegenüber. Eine genauere Analyse des neuronalen Substrats schließt sich mit Blick auf die Haupteigenschaften von CSPs an. Am Ende wird ein Überblick über Anwendungsmöglichkeiten in der Robotik gegeben, der begleitet wird von einer Fallstudie.

Contents

Chapter 1

Motivation

Constraint Satisfaction Problems are ubiquitous in the daily life of humans. One simple example is the planning of how to get from a certain location A to another location B with a city map. Robots are designed to simplify the human life and therefore, face the same challenges as humans. Their problems have to be treated as CSPs as well in order to find a good solution.
One robotic task is the aforementioned path planning. But also the exploration of a network is in the field of duty. Path planning can involve one or more robots increasing the complexity of the task at the same time. The idea in both cases is to divide the real environment into small cells. These are the variables of the problem. The robot operating in the network has to find the best possible direction for a future move to fulfill its task. During the process it is hindered by walls or other robots in some directions just like a human in the city meeting other people. At each iteration step it calculates new data for its mission depending on the existing information. Eventually, it also communicates with other involved robots.
It is of great interest to fulfill the task most efficiently and accurately as this saves energy and time and avoids damages to the robot and the environment. With other words the robot shall achieve the best possible result. Therefore, the calculation method has to work conveniently in order to be able to navigate successfully and quickly through the network.
Different approaches which are adequate for this task have to be compared. A wide range of problem solvers from classical approaches to neural networks is known. Classical approaches look for the solution sequentially while assigning one value to one variable at a time step. This is why it is taking exponential time to solve the problem. Neural networks see a lot of attention recently as they appear to be very efficient with CSPs. Their implementation as problem solvers for robotic tasks is of great interest in this report. While studying functionalities, differences and advantages of the methods presented below the best can be selected.
With new powerful processors good results can be achieved in determining the best path for the robot.

Chapter 2

Constraint Satisfaction Basics

A Constraint Satisfaction Problem (CSP) is made up of three sets: the Variables $X_i = X_1...X_n$ describing the state of the system, the Domains $D_i = D_1...D_n$ providing the possible values of each Variable X_i and the Constraints $C_j = C_1...C_m$ restricting the value assignment to each variable it involves. The aim is to find a value for each variable such that all constraints can be fulfilled. This is called a complete solution.
A CSP is graphically represented by a constraint graph. The nodes of the graph are the variables. They are linked to each other to represent the constraints between the variables.
To estimate the closeness to a solution it is necessary to know how many constraints are violated in a given state. The sum of all violated constraints is called the energy of state a or cost of an assignment:

$$E = \sum_a C_a(x_a) \tag{2.1}$$

C_a is 1 if the constraint is violated, otherwise it is 0. The goal is to minimize the energy function. During this process it is possible that the minimization gets stuck in a local minima which seems to be the best solution in the surround but not the overall solution. In Fig. 2.1 point A is a local minimum whereas point B is the preferred solution. Therefore the process has to be triggered to overcome local minima and hence find the global minimum. This can be done for example by random movement to higher energy states or by assigning weights to the constraints.
Concerning the number of involved variables there are different types of constraints. The unary constraint limits only the assignment of values to one variable, e.g. $x_1 = 3$. A constraint like $x_1 \neq x_2$ which involves two variables is called a binary constraint. Constraints limiting the assignment of values to three or more variables are called higher order constraints, e.g. $x_1 \neq x_2 \neq x_3$. Those are very hard to solve and often have to be broken down into binary constraints: $x_1 \neq x_2$, $x_1 \neq x_3$, $x_2 \neq x_3$.
In neural networks (NN) the variables of the CSP are built up by neural units with different approaches to process the input. The constraints are modeled through

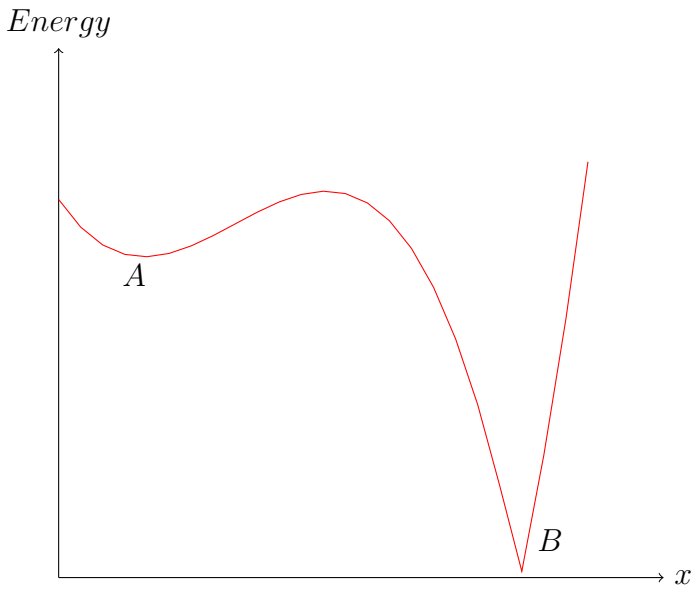

Figure 2.1: A is a local minimum of the cost function

weighted interconnections of those neural units. Therefore, the energy function is dependent on the connection strength and the input of other neurons.

$$E = (\sum_{i<j} w_{ij} A_j + \sum_{i} \Theta_i) \tag{2.2}$$

A_j is the state of the jth neuron connected to neuron i and Θ_i is an activation threshold of the neuron i. The neural network can overcome local minima by noise (simulated annealing) or phase minima (see Chapter 3). Because of the binary links between two neural units normally only unary and binary constraints can be implemented directly. For higher order constraints intermediate variables are necessary.
The details of neural networks, their functionality and their approach to CSP are presented in the upcoming chapter.

Chapter 3

Neural Networks

3.1 Constraint Satisfaction Adaptive Neural Network

As described by Yang et al. in [YW00] a constraint satisfaction adaptive neural network (CSANN) is made up by interconnected parallel processing elements. These neural units compute local information which is transmitted via connections to other neural units. One unit consists of two serialized parts: a linear summator and a nonlinear activation function (see Fig. 3.1).
The summator receives all activations from the connected units. Those values are weighted according to the connection weight and then summed up with a bias current.

$$N_i = \sum_{j=1}^{n} (W_{ij} A_j) + B_i \tag{3.1}$$

The resulting value is input to the unit activation function which depends on the configuration of the network. There are three different types of activation functions proposed in [YW00]: a linear threshold function, a linear-segmented function and a S-shaped function. The output of the activation function is called activation of the neural unit and is also a new input value for another neural unit.

$$A_i = f(N_i) \tag{3.2}$$

The modulation of the constraints is done by a dedicated activation function depending on the net input. Therefore, it is necessary to have different types of neurons to describe different types of constraints. In a more complex CSP like the job-shop scheduling problem the constraints have to be separated into different groups and are modeled differently. Afterwards they have to be linked according to every neurons' required input. The unit can provide output for the CSP or only intermediate values for other units.

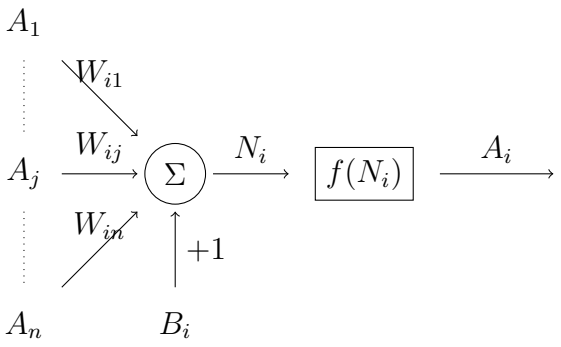

Figure 3.1: A neural unit of the CSANN, [YW00]

3.2 Boltzmann Machine

A Boltzmann Machine is a neural network with stochastic input/output functions of its units. According to Hinton et al. in [HSA84] the neural unit adopts the states according to a probabilistic function which is dependent on the state of its neighboring units. The input that it gets from its neighbors is also dependent on the weights on their links. The connection weights are symmetric (see [SAS90]) and real-valued. They can change according to the probability of the state. The net input of the ith unit at the instant k is

$$net_i(k) = \sum_j w_{ij} s_j(k) - \tau_i \tag{3.3}$$

with w_{ij}: connection weight and τ_i: threshold of unit i.
A unit can only be in two binary states: On and Off. On means that the unit accepts the hypothesis provided by the other units. Off signals a rejection of the hypothesis.
As aforementioned the unit output depends stochastically on the input. The probablity of unit i being in the On state is

$$p_i = Prob[s_i(k+1) = 1] = \frac{1}{1 + exp(-\beta net_i(k))} \tag{3.4}$$

The presented network shows similarities to the Hopfield Neural Network (HNN), but the HNN uses a threshold function to calculate its output.
The constraints are modulated by the weight on the link. It represents a binary constraint of two hypothesis. A positive weight stands for a support between the two hypothesis leading to a higher probability that the hypothesis will be accepted. A negative weight signals the opposite case for the hypothesis. Due to the binary links the network can only model binary constraints. Higher-order constraints have to be modeled with intermediate variables.
Since the energy of a global state α: $S(\alpha) = [s_i(\alpha), s_j(\alpha), \dots]$ is dependent on the

constraint violation, it is also dependent on the connection weight:

$$E_\alpha = -\frac{1}{2}\sum_i \sum jw_{ij}s_i^\alpha s_j^\alpha + \sum_i s_i^\alpha \tau_i \tag{3.5}$$

To overcome local minima the Boltzmann Machines use simulated annealing according to [SAS90]. A higher temperature is introduced to the network and decreased with time (cooling process). At high temperature the state changes occur often, at lower temperature a favorable solution will be biased strongly.
A great advantage of Boltzmann Machines is that they learn and adapt the weights throughout the search for a solution of the CSP (see chapter 5.1).

3.3 Limit Cyclic Attractors

Limit Cyclic Attractors as presented by Mostafa et al. make up a "a population-level, rate-based network" [MMI13]. A linear threshold unit (LTU) with the following dynamics is the model for the population:

$$\tau_i \dot{x_i}(t) + x_i(t) = max(0, \sum_j w_{ij}x_j(t) - T_i) \tag{3.6}$$

x_i is the firing rate, w_{ij} the connection weight of unit j and i, τ_i and T_i are the time constant and the unit threshold respectively.
One limit cyclic attractor is built up by asymmetrically coupling Winner-take-all (WTA) circuits in a loop as displayed by Fig. 3.2. In a WTA circuit the cell with the largest initial value is the winner of the network (compare [SN93]). The output of all other cells in the network goes down to 0.

This configuration shown in Fig. 3.2 helps to destroy fixed point attractors in each WTA stage as the loop connection causes the network to exhibit oscillatory activity. A fixed point attractor is shown in Fig. 3.3. The plastic connections in the limit cyclic attractor obey a learning rule which is analogous to the Bienenstock-Cooper-Munro (BCM) rule (see chapter 5.1). If an input pulse is applied to the bottom WTA stage, it starts oscillatory activity. Due to the connections a bump of activity continuously jumps from one WTA stage to the next. At the same time the inhibitory population will shut down activity in the lower stage. Through the plastic connection the highest population will excite the bottom stage according the connection weight. Without external input the population favors the population with the stronger projection weight.
The variables of the CSP are represented by one network as shown in 3.2. This network will only have binary variables. The domain can be enlarged by more excitatory populations in the bottom WTA stage.
Binary constraints can easily be modeled by coupling the bottom stage excitatory populations of two variables. Higher-order constraints can only be modeled by using intermediate networks with a higher cardinality (i.e. a larger number of bottom stage

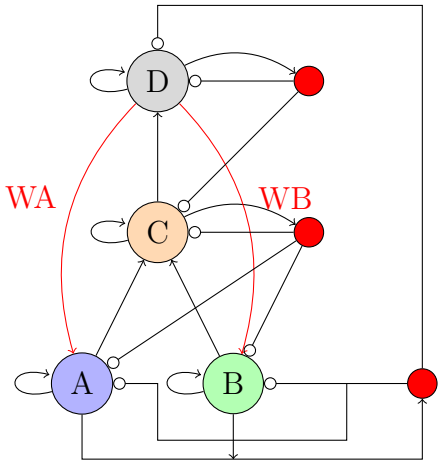

Figure 3.2: A Limit Cyclic Attractor cell as described by [MMI13]. The red arrows symbolize the plastic connections obeying the BCM rule. The red units are the inhibitory units of the level.

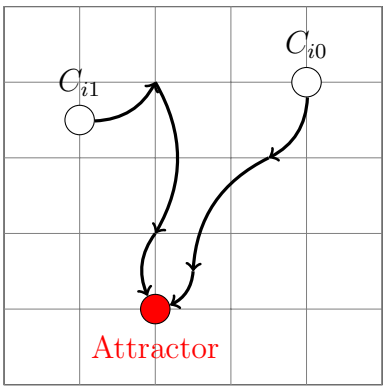

Figure 3.3: A fixed point attractor always attracts the variables' values towards itself. C_i is the initial condition of the system.

populations).
In contrast to the neural networks the classical approaches solve the problem rather sequentially than simultaneously. Ideas to accelerate the search by parallelization exist. The following chapter is providing a more detailed look on this aspect of CSP solving.

Chapter 4

Algorithms, Heuristics and Implementation Details

4.1 Algorithms

Many algorithms have been developed until today. Therefore, only the frequently used ones will be presented in the following.

Backtracking This basic approach mentioned in [RN02] assigns one value to one variable at a certain time and check for constraint violations. If the value is non-compliant with any constraint the algorithm backtracks and assigns a new value to the variable. If there are no possible values in the domain which comply to a constraint it backtracks even further. The backtracking algorithm is inefficient according to [BPS99].

Local Search During the initial assignment every variable is assigned a random value of its domain. During the following steps the value of one variable is changed at a time. This change is searching to minimize the constraint violation. The Local Search Algorithm is very effective [RN02].

Forward Checking In contrast to the backtracking algorithm the forward checking approach does not only check the past and current variables for constraint violation but also deletes (temporarily) all values from future variables from their domain when they are not compliant to the newly assigned value (compare [BPS99]).

Maintaining Arc Consistency (MAC) The MAC algorithm even goes a step further by checking also future variables against each other. It also deletes (temporarily) the values of future variables which are mutually exclusive and those which are not compliant to the current assignment. As stated in [BPS99] the hope is that the extra time invested on future assignment leads to an overall saving of computation time.

4.2 Heuristics

Heuristics help to chose a start variable or a variable with which to proceed the search. Since there are a lot of heuristics known today, only the most popular ones will be presented in the following.

Static Ordering of Variables The order of the variables during the search is defined at the beginning of the search.

Dynamic Ordering of Variables The choice which variable is selected next is done at each new iteration of the search and is dependent on the current state. However, this approach is not feasible for all algorithms according to [BPS99] as some lack information about the future variables.

Minimum remaining value (MRV) This heuristic is also known as fail-first, most constrained variable heuristic according to [RN02]. It looks for the variable with the smallest current domain. This avoids pointless searches which in the end all fail.

Degree heuristic This is a good heuristic when looking for a starting point of the search (compare [RN02]). It choses the variable which is involved in most constraints or with other words which nodes have the most arc connections.

Least-constraining This one is also known as the minimum conflict heuristic as it always prefers the values of a domain which leave the most freedom for the other variables [RN02].

4.3 Implementation Details

Parallelization Parallelization is necessary for solving CSP if the number of variables increases. There are two approaches to parallelize the search.
The first one uses every available processor to perform a sequential search (compare [CCDA11]). If a processor has found a valid solution it sends a "Solution found" message. After a dedicated amount of iteration steps m the system checks for solution messages. If such a message exists the search is terminated. The process can be improved by checking every mth iteration step also for a more mature solution (i.e. lower energy value) of other processors. If such a solution exists, the processors farther away from the solution reset their search randomly. For example, in Fig. 4.1 (a) Processor 1 and 2 would reset their search after the first iteration step as their energy function have a higher value than the one of Processor 3.
The second approach presented by Regin et al. in [RRM13] divides the problem into subproblems. The trick is to generate a lot more subproblems than processors. Every processor takes up a new subproblem when finished according to the FIFO principle (i.e. the problems are queued in the order of their

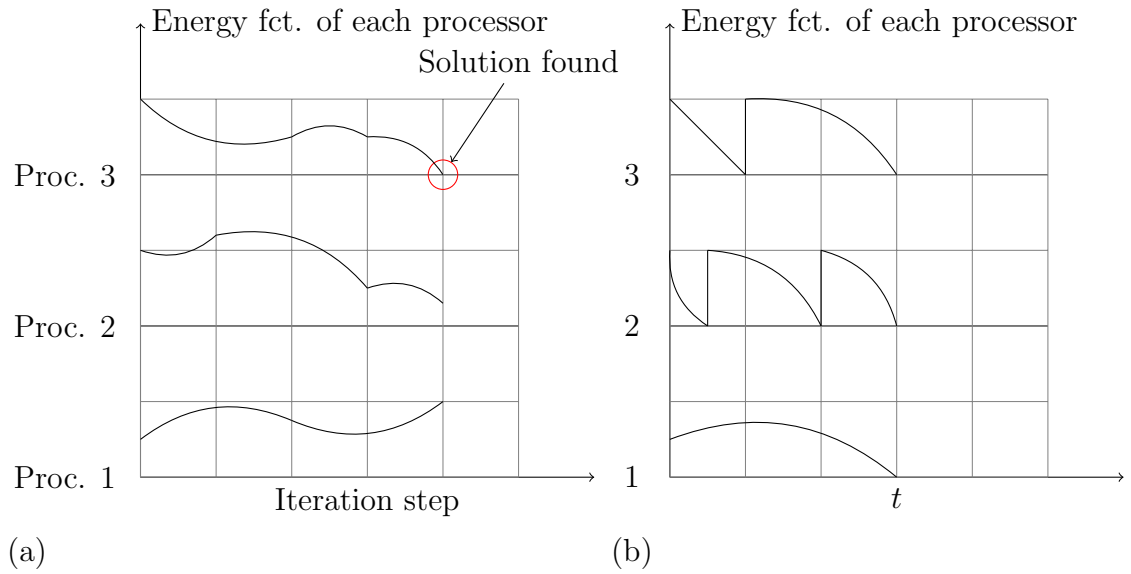

Figure 4.1: (a) Every processor performs the search individually. As soon as a solution is found the search is exited. (b) The problem is divided into subproblems. Each processor solves a subproblem and takes up a new one. When all subproblems (in this case 6) are solved the search finishes.

definition and solved in the same order). This way it can be guaranteed that the time for the complete solution is approximately equally divided among the processors (compare Fig. 4.1 (b)). Finally, the subsolutions only have to be compiled to generate the full solution. Therefore, the resolution time decreases.

Distributed CSP In contrast to parallelization which is concentrating on increasing the efficiency of the search, distributed CSP (DCSP) deals with the complexity of centralizing the information. According to [YDIK98] one reason is that it is sometimes too costly to centralize all constraints and variables in one machine. Another one is that it can be impossible to gather all information due to security reasons.
A DCSP is realized with a multi-agent system. In general, each (automated) agent is assigned to one variable and is the only one that knows the full set of possible values. The constraints are the links between the agents.
An agent is a physical, virtual identity that lives and acts in an environment, senses it, has a reactive behavior and is driven by certain purposes (compare [MMH13]). The agents communicate with each other by sending messages.
Different implementations exist: Moghaddas et al. describe a multi-agent community in [MMH13]. Each agent has a score expressing the negative value of the violated constraints (comparable to an energy function). The agents store and calculate their own scores and try to maximize them. Every agent

stores a list of other agents to which it is not compliant. This is called the CSCommunity. In this community the agent which created it is the leader. The agents process "exchange value" operations among each other if that is beneficial for at least one of them until all scores are 0. In Fig. 4.2 it is clear that if Agent A changes it value to 0 the scores of both agents can be reduced to 0.

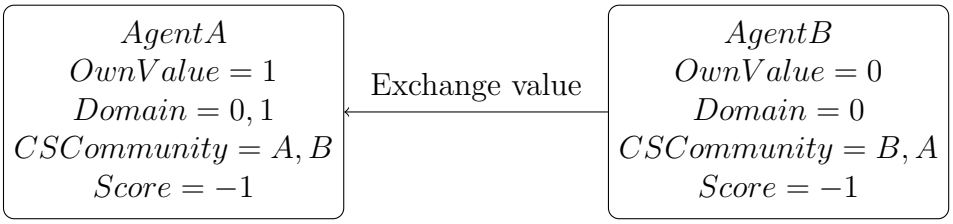

Figure 4.2: Approach presented by Moghaddas et al. in [MMH13]. The constraint between the variable of A and B is $Variable_A = Variable_B$.

In [YDIK98] Yokoo et al. present a different approach with binary constraints as directed links. One agent is the value-sending agent which transmits an "ok?"-message with the assigned value to the linked agent. This one – the evaluating agent – checks for the compliance of the current value with its own value. If it discovers a noncompliance it changes its value if possible. Otherwise it replies with a "nogood"-message (compare Fig. 4.3) and the value-sending agent has to assign a new value. To overcome cycles of directed links the agents get prioritized. This approach is called Asynchronous Backtracking.
An improvement of the Asynchronous Backtracking is the Asynchronous Weak Commitment Search [YDIK98]. The agents record the "nogood"-sets. If there is no consistent solution, the existing partial solutions are abandoned and the search is restarted. The agents also keep track on related agents called neighbors. Among them the priorities become adapted to the inconsistency of the partial solution. The priority is also part of the "ok?"-message to signal it to the other evaluating agent.
The messages sent between the agents contain information about its constraints which influence the problem and the values which might be suitable. But the domain of an agent may be a lot larger but is not presented to the other systems as they are not interesting for the problem. Therefore, the presented approaches can be applied to problems with higher security restrictions.

Several ideas have been developed to advance the search for a solution. This has not only improved the neural networks but also the classical search. These ideas are presented in the next chapter.

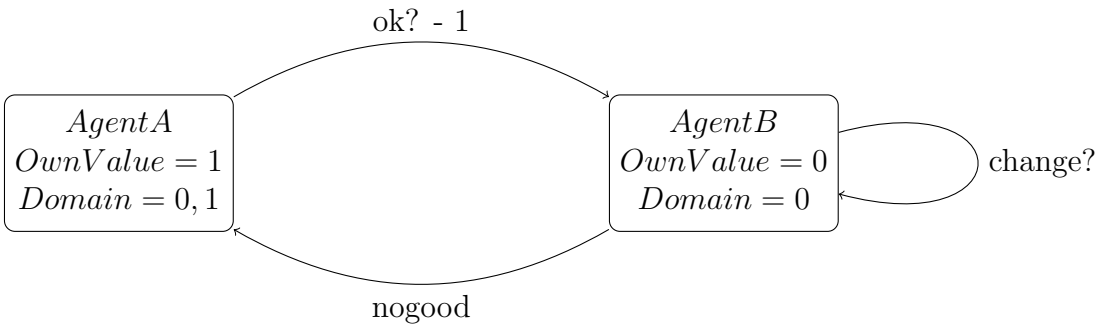

Figure 4.3: Approach presented by Yokoo et al. in [YDIK98]. B is not able to change its value and sends a "nogood"-message. Now it is up to agent A to assign a new value. The constraint between the variable of A and B is $Variable_A = Variable_B$ like in Fig. 4.2.

Chapter 5

Adaptation Mechanisms and Optimisation in CSPs

5.1 Learning

During learning the system becomes trained on certain input data and generalizes their relation. Afterwards, it is able to apply the knowledge on new data. In neural networks this learning process is done by adapting the connection weights of the units. Two of the presented neural networks from chapter 3 are able to learn. This is a key feature of both approaches for solving a CSP as the speed of search can be accelerated.

5.1.1 Boltzmann Machines

In a Boltzmann Machine the connection strength between the units is modified. The connection strength models the acceptance of a hypothesis. This modification is based on the probability of a unit being in a state α:

$$\frac{\partial \ln P_\alpha}{\partial w_{ij}} = \frac{1}{T}[s_i^\alpha s_j^\alpha - p'_{ij}] \tag{5.1}$$

The probability is dependent on the temperature T, the value of the units i, j in state α and the probability of i and j being On at the same time. Formula 5.1 only accounts for thermal equilibrium (i.e. after the simulated annealing).
If the environment directly specifies P_α it is quiet easy to derive the correspondent weights. Generally, this does not occur very often as the environment does not specify all the important information. The constraints – provided by the environment – normally are of higher-order. Therefore, an internal representation is required. In case of [HSA84] hidden units were implemented to break down these constraints into binary constraints. However, the environment only sees the visible units.
We define $P'(V_\alpha)$ as the probability of the free running network and $P(V_\alpha)$ as the

probability of the determined network. The discrepancy between both is a measure of the distance to an adapted system. It shall be 0 if both are identical.

$$G = \sum_{\alpha} P(V_\alpha) \ln \frac{P(V_\alpha)}{P'(V_\alpha)} \tag{5.2}$$

As $P'(V_\alpha)$ is dependent on the connection weights, the discrepancy can be minimized by altering the weights. The formula 5.2 is sometimes called information gain. To perform a descent in G it is necessary to know the dependence of G and w_{ij}:

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}(p_{ij} - p'_{ij}) \tag{5.3}$$

p_{ij} is the average possibility of the two units i, j being On for the determined network. p'_{ij} is the corresponding probability for the freely running network.
By changing the weights by an amount related to the the difference of the probabilities and a scale factor G can be minimized.

$$\Delta w_{ij} = \varepsilon(p_{ij} - p'_{ij}) \tag{5.4}$$

All the probabilities have to be measured at thermal equilibrium as well. The learning can be controlled via the scale factor ε to reduce the effect of the noise or the collection of statistics has to be done for a longer period.
An advantage of this method is that it only depends on two units but optimizes the system globally. In CSP solving the learning of the Boltzmann Machines improves the search. As the weights represent the support or rejection of a hypothesis (and therefore, a constraint) a change upon the probabilities means an understanding of the constraints by the network. The adapting weights enable the network to solve the problem more mature.

5.1.2 Limit Cyclic Attractors

As described by chapter 3.3 the top WTA stage is connected to the down stage via plastic connectors. These connectors obey a learning rule analogous to the Bienenstock-Cooper-Munro (BCM) rule:

$$\dot{w}(t) = Ku(t)\left(\frac{(w(t) - w_{min})[v_{th} - v(t)]^-}{\tau_{dep}} + \frac{(w_{max} - w(t))[v(t) - v_{th}]^+}{\tau_{pot}}\right) \tag{5.5}$$

$[x]^+ = max(0, x)$ and $[x]^- = min(0, x)$. $w(t)$ is called the connection weight. $u(t)$ is the activity of the source population whereas $v(t)$ is the activity of the target population and v_{th} is threshold of the target population which delimits the transition between potentiation and depression. w_{min} and w_{max} are the soft bounds on the weight. τ_{dep} and τ_{pot} are the depression time constant and potentiation time constant respectively. K is called the plasticity rate.

The BCM rule is a further development of the Hebbian Learning rule which says that the connection weight of two cells activated at the same time increases ([Heb59]). The BCM theory was first published as a model of learning in the visual cortex but proved to be very realistic in other fields of brain learning as well. In contrast to the Hebbian Learning it does not only include a theory of the potentiation but also a theory of the depression of the connector strength (see [BCM82]).
Lower activities between two neurons or a neuron's failure to activate another neuron will decrease their connection weight. This effect is called Long-term depression (LTD). In the opposite case when a neuron is often part in the activation of another neuron the connection weight will increase – this is called Long-term potentiation (LTP) (compare Fig. 5.1). Therefore, the connection weights in the Limit Cyclic Attractor adapt the firing rate behavior of the populations and select a winner in the bottom WTA stage. In CSP this enables the network e.g. to select variables which exclude each other. Due to this effect the network will solve the problem faster than other approaches without a learning method.

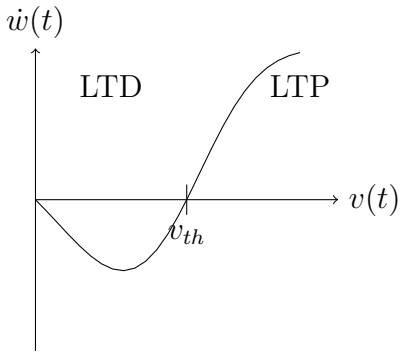

Figure 5.1: The weight adaptation is dependent on the output activity, [CB08]

5.2 Interchangeability

According to Neagu in [NA00] several forms of interchangeability are known:

- Full Interchangeability: The exchange of two values for a variable requires no further changes as the solution stays valid.
- Neighborhood Interchangeability: It implies full interchangeability because the two values exchanged are compliant to every constraint.
- Partial Interchangeability: An exchange in one variable may cause further changes in the dependent set to keep the solution valid.
- Neighborhood Partial Interchangeability (NPI): It is a weaker form of neighborhood interchangeability. It concentrates on a subset of variables and its

neighborhood. Every constraint between the variable in the set and the neighborhood allows an exchange in that variable.

The dependent set S is the set of variables which also has to change so that the solution remains consistent during an exchange of one variable. A minimal dependent set requires that no subset of S is also a dependent set for the interchangeability. A minimum dependent set does not allow the existence of another dependent set with a smaller cardinality.
Processing an exchange of (neighborhood) partial interchangeable values means exchanging values of other variables in S. Therefore, it is more costly than processing an exchange in neighborhood interchangeability. It is still worth the effort when it is possible to enhance the search and backtracking or adapt an existing solution easily to new conditions.
When a CSP has been completely solved and a new problem arrives in the system the problem solving process can look like this according to Neagu in [NF01]: The problem is compared with a previous case and the adaptation process receives the following inputs: the solution of the retrieved case, the corresponding CSP of the new problem and the differences between both problems. Afterwards an algorithm is applied that looks for NPI in the system. The algorithm is described in [NF01].
A found match for all variables is a solution to the new problem. This process is very efficient compared to solving the CSP completely again. If no such solution can be found the problem solving can still restart from the scratch. In daily life this can happen frequently as most complex CSPs have to be solved once and afterwards the parameters change only little over the process time.
In the following chapter the similarities and differences of the aforementioned methods shall be discussed with focus on their utility for CSP.

Chapter 6

Discussion

Considering daily problems as CSPs shows advantages as this method is very similar to the way humans deal with them. Therefore, the treatment sees a lot of attention. The classical approaches involve algorithms which search sequentially for values for the variables. Some algorithm as the Forward Checking algorithm even try to "predict" the values of future variables. However, most algorithms assign one value to one variable at a time. The goal is to reduce the energy of the state, i.e. reduce the number of violated constraints by the current assignment. In classical approaches the constraint representation does not happen via connections but exists as mathematical formulation. Hence, the strengths of the constraint cannot be modeled. The algorithms are not able to learn either.
In general, CSPs are known to be NP-complete according to Kasif et al. in [KD94]. With the complexity of the problem the time to solve it rises. Sometimes they even get stuck or take a too long time to find a solution. Therefore, heuristics are implemented to simplify the model beforehand and thus avoid getting stuck. Several heuristics are known for different task, e.g. choose a variable to start with or select a variable to proceed with.
As aforementioned the complete search for a solution can be very time-consuming. In order to re-use existing solutions and to save time when the constraints vary little from the formerly existing ones the concept of interchangeability is introduced. The algorithm looks for interchangeable values in the current solution and the domain of the variables. Then it adapts the current solution to the new conditions by interchanging values.
To accelerate the search parallelization is inevitable. This allows the user to reduce the time for the system to find a solution or find more than one possible solution at a time. With the powerful processors developed lately it is possible to reduce the complexity of the problem significantly (compare [KD94]). An important development in the parallel approaches is the distributed CSP as it allows to save some information only to one unit called agent. This is very important for applications with security restrictions. The agent communicate and interact with each other, sense their environment and are driven on certain purposes, e.g. find the best value

for the CSP among their domain.
With the development of neural networks – simplified models of the human brain – new possibilities came up. Such a network solves the problem simultaneously and considers all variables at once. This leads faster to a solution. A NN consists of neural units which represent the variables. They are connected by weighted links between each other. These weighted connections represent the constraints between the variables.
The output of one unit is classically dependent on the weighted input from other units and the activation function of the neuron. This output is at the same time input for another neuron in the network.
The CSANN uses the weighted and summed input of the connected neurons as the input of its activation function. It can have different dependences (e.g. a linear threshold function). The output is the neuron's excitation. In contrast to CSANN the Boltzmann Machine's activation is stochastically dependent on the summed and weighted input. Additionally, the Boltzmann Machines are able to learn. The learning is represented by the weight adaptation. This happens throughout the process according to the differences of the state probabilities between the determined and the free running network. Another network that is learning is the network of Limit Cyclic Attractors. Several WTA circuits are coupled asymmetrically. The excitation spreads from the bottom level to the upper levels which inhibit the activation in the stages below. The highest stage is connected with a plasticity connector to the bottom stage. This plasticity connection obeys an analogous BCM rule. Hence, during the process the weights on the connection change according to the firing rates of the populations. This represents the learning process. The circuit at the bottom stage with the better connection weight "wins".
The learning of relations between the variables can be considered a key feature of the NNs. As a consequence, the network solves the problems even faster and more mature.
However, with those networks only unary and binary constraints can be implemented as the binary links do not allow the modeling of higher-order constraints. Whenever higher-order constraints are involved they have to be broken down into lower-order constraints via inter-mediate units. During this process the environment can also be simplified by classical heuristics beforehand to reduce the complexity of the problem. However, the fact that no higher-order constraints can be implemented can lead to a huge amount of neural units in a network. Another point is that sometimes as NNs are only proven under limited research conditions. For real problems sometimes human expertise is necessary to tune the final values.

Among the classical approaches and the neural networks the similarities between the distributed CSP and the neural unit comes to mind. In both cases the problem is solved by units and agents respectively. They are linked to each other. They neural units are linked with weighted connections whereas the agents are linked without a weighted connection. Therefore, they are both able to communicate with

each other. While communicating they are updating their information about the current status of the involved partners. A difference is that neural units can do this instantly whereas agents do this only iteratively at a dedicated point in time.
The agents try to minimize the energy of the state by changing their values. They communicate this change at each iteration step and calculate the new energy value. Their communication basis does not change during the process.
In the neural model the units influence each other with their firing rates. They also adapt their weights according to the firing rates or the probability of the state that they are in. This leads to changes in the connectivity among the agents. These changes then define the global solution.
To prove the applicability of the CSP formalization on existing situations several application scenarios are presented below.

Chapter 7

Application Scenarios

Constraint Satisfaction Problems occur in daily life. Therefore, there is a wide field of possible applications like Localization, Job Shop Scheduling, Car Sequencing, Cutting Stock, Vehicle Routing, Timetabling, Rostering, Resource Allocation, Robotics. This paper concentrates on the application in the robotics sector. Three different robotic applications of CSP shall be discussed. The path planning in a network for one robot and its extension to a multi-robot environment and the exploration of a network by several robots.

Robot Path Planning The aim is to find a collision free path for the robot along which it moves from a start point to a target point. Therefore, a grid of small cells divides the real environment. The size of cells defines the resolution of the network according to [ST02]. The robot's memory stores its current x-coordinate, its current y-coordinate and the occupation attribute of the cell. The occupation attribute is a measure for the occupation of the cell. It can be crisply defined with binary states: 0,1. 0 signalizes the robot that the cell is free, 1 signalizes that it is occupied. The occupation attribute can also depend on a linear function for example – the closer to the object the higher the occupation attribute. This way also fuzzy environments can be modeled.
The variables of the CSP are the position of the robot with a domain of 8 possible directions (similar to the main directions on a map: North, Northeast, East, Southeast, South, Southwest, West, Northwest) (compare also Fig. 7.1). Two important constraints are applicable here. Firstly, the robot must not move in a direction which is occupied. Secondly, it shall move in a direction closer to the target.
The problem is constructed such that a position farther away from the target rises the energy of this position. Therefore, the goal is to minimize the energy function (see chapter 2). The energy function of this scenario is dependent on the distance to the target and an uncertainty factor (see [ST02]):

$$E(x,y) = \frac{1}{2} * w_1 * (x_{current} - x_{target})^2 + \frac{1}{2} * w_2 * (y_{current} - y_{target})^2 + I * uncertainty(x,y) \tag{7.1}$$

The search procedure then consists of 3 basic steps: 1.) Deletion of directions to cells with obstacles from the domain, 2.) Calculation of the energy of neighbor cells via a neural network or an optimizer, 3.) Movement towards the lowest energy value. This procedure repeats itself until the target is reached, i.e. the energy value is 0.
Attractors, e.g. dead end paths, may appear in this scenario. To avoid getting stuck in them it is important to choose the right weights, avoid repeated moves and save local minima. Another good point is to predict the next moves of the robot regarding to a predefined depth. If this depth is too high a lot of cells will be deleted beforehand. By decreasing the prediction depth more cells will appear in the domain.
By changing the planning direction from "start to target" to "target to start" differences in planned paths become visible. To achieve the best result both strategies can be applied and the best points of both become combined in one plan.

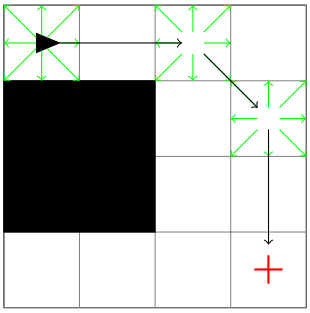

Figure 7.1: A robot path planning approach: the triangle marks the start position of the robot. The red plus sign signals the target location. The green arrows point in the directions, the black arrows show the chosen direction.

Multi-robot Path Planning With several robots in a network it is necessary to leave the organization to a centralized planner. It implements priorities for the robots (see [Rya10]). The collection of robots is treated as a single entity. In this case the map is also split into subgraphs like in the aforementioned procedure. In the scenario described in by Ryan in [Rya10] two different types of subgraphs are used: cliques and halls. The cliques are large open spaces whereas halls are smaller corridors preventing overtaking.
In this case the variables are slightly different. Each robot has a set of three variables: the index of the subgraph it is located in, the index of the first vertex it occupies and the index of the last vertex it covers.
Besides the constraints mentioned above for the robot path planning (no moves into occupied directions, always move closer to the target) there are four other constraints which constrain the behavior of the robots in the subgraphs and among each other. Firstly, the robots are only allowed to move between neigh-

boring cells. Secondly, the first and last vertex of one robot have to be in the same subgraph. Thirdly, the same vertex cannot be covered by two robots at the same time. Finally, robots are ordered if they are in the same hall (i.e. no overtaking is allowed).
The search is processed by a constraint solver which also applies heuristics.
The results mentioned by Ryan in [Rya10] show that the robots navigate through the network rather quick. It still depends on the applied heuristics, the priorities and the abstraction level.

Exploration of a Network The task of the robots is to explore a network [DBD+09].
Therefore, they collaborate and spread out exchanging their partial maps to achieve a global map.
The variables in this scenario are also the position of the robot and the domain contains 8 possible directions (see Robot Path Planning and Fig. 7.2). The robots still have to measure their communication range to the next robot and their sensor range.
The constraints are different to those of the path planning approaches as the robots try to find as much information about the network as possible. Firstly, the future location must not break the connectivity of the network. Secondly, the future location does not lead to an overlap of the sensors. Violating this constraint would lead to a decrease of exploration speed. Finally, the future location has to be nearest possible to current frontier. This constraint shall also accelerate the exploration.
The search procedure consists of four steps: 1.) Update of the maps and connectivity tables for each robot, 2.) Construction of the CSP based on updated data, 3.) Ordering of the values in each domain according to the distance to frontier, 4.) Solving of the CSP to obtain future directions of each robot.

In [DBD+09] Doniec et al. also present results of a simulation. They state that an environment is fast explored and adding robots to a certain limit increases the speed of the exploration. Nevertheless, when too many robots are located in the network they become more occupied by avoiding each other than exploring the network.

The robotic task of navigating through a network – which is also part of the exploration of a network – shall be combined with a neural network as a problem solver in the following case study.

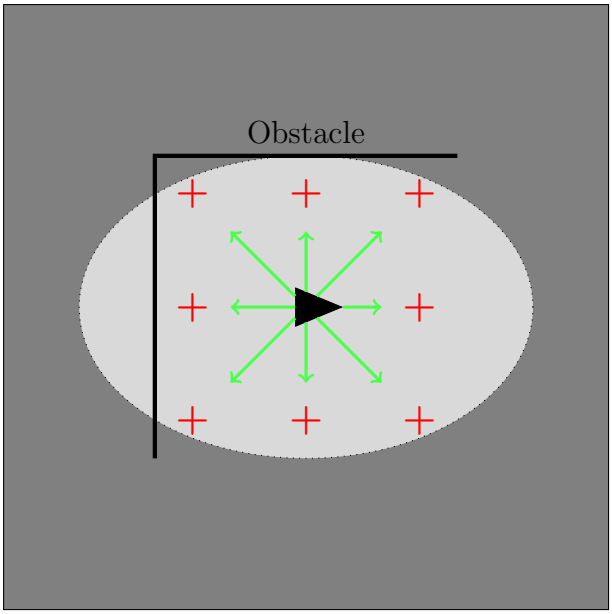

Figure 7.2: The principle approach of Doniec et al. in [DBD+09]. The black triangle signals the current position of one robot. The red plus-signs stand for the 8 future positions and the green arrows mark their directions. The light grey space is the explored area whereas the dark grey area is the unexplored one. The dotted line marks the frontier.

Chapter 8

Case Study

This case study shall provide an example of an application scenario mentioned in chapter 7. The robot's task is to navigate in a network. It is neither allowed to move trough the obstacles nor jump over them. It possesses two sensors: vision and ultrasonic.
Three different test cases are evaluated. In the first one (A) the robot can use both sensors. The second test case (B) is done during night – the robot cannot relay upon the vision. The last test case (C) is done with walls that reflect.
The formal definition of the CSP:

Variables The position of the robot in the network $q = \begin{pmatrix} x & y & t \end{pmatrix}^T$

Domains 8 Directions at a time t: $\begin{pmatrix} \pm x & \pm y & 1 \end{pmatrix}^T$, $\begin{pmatrix} \pm x & 0 & 1 \end{pmatrix}^T$ and $\begin{pmatrix} 0 & \pm y & 1 \end{pmatrix}^T$

Constraints To avoid the robot crashing into the wall the distance between the wall and the robot must not fall below a certain distance, e.g. 1m.

$$\frac{\partial q}{\partial t} \leq distance_{robot-wall,min} \tag{8.1}$$

Or with other words: the future position has to be lower than the act. distance to wall

$$q_{i+1} \leq distance_{robot-wall,act} \tag{8.2}$$

The values for equation 8.1 and 8.2 derive from the change of the position over the time

$$\frac{\partial q}{\partial t} = \begin{pmatrix} \pm c_x \\ \pm c_y \\ 1 \end{pmatrix} q + offset \tag{8.3}$$

$$q_{i+1} = q_i + \frac{\partial q}{\partial t} \tag{8.4}$$

It can easily be seen that the constraints are dependent on the sensors' input. With a decrease of the distance measured by the sensors the strength of the constraint increases.

Implementation A neural network shall calculate the future direction of the robot considering the desirability of the eight possible directions. As described in chapter 3.3 within the Limit Cyclic Attractor network it is possible to enlarge the domain of a variable by adding WTA circuits to the bottom stage. This is why this network shall be the model network for this case study. The number of bottom stage WTA circuits represents the size of the domain. In this case the size is eight. Therefore, eight bottom stage circuits are necessary to process the information. They are working simultaneously. Each neuron is responsible for one direction.

The bottom stage neuron gets the summed input of a bias current and the distance measured by the sensors. This is the distance to the wall in its assigned direction. If the distance to the wall in one direction is very small the sum is small as well. Otherwise, the input is large. This will lead to different firing rates of the neurons. Hence, it is easy to determine directions which contain no obstacles.

When the robot has to find a specific target coordinate a current for a favorable direction can be added as described by Horiuchi in [Hor06]. Otherwise, this current is 0 for all neurons. When the sum overcomes a certain threshold the neuron starts firing.

The middle level WTA circuit (compare Fig. 3.2) becomes excited by the bottom stage neurons and excites the upper stage neuron similarly. The plastic connectors described in chapter 5.1 will then excite the bottom neuron with the highest excitatory rate as its connection weight is dependent on the firing rate of the target population. This process obeys the BCM rule and is called Long Term Potentiation. At the same time the other neurons will be inhibited by this connection (Long Term Depression). With other words due to plastic connections obeying the BCM rule the WTA circuit with the largest input is selected and its dedicated direction is considered as the most desirable (compare Fig. 8.1). Due to the inhibition of all other firing units this allows the network to react and to calculate quickly the necessary information. The "winning" direction is input for the robots actuator unit.

This kind of network is also described by Horiuchi in [Hor06] for a representation of bat navigation.

Sensors In the test case (A) both sensors work. Their measurements have to be compared to find congruence among the data. Afterwards, the result is fed to the neural network. In the test cases (B) and (C) with hindered conditions the data of at least one sensor is useless. Therefore, certain thresholds have to be introduced to enhance the evaluation. In (B) the sensor will only deliver a very small output as the environment is dark. A bottom threshold is necessary to allow only stronger signals to pass. In this case the robot can only rely on the data of the ultrasonic sensor. With shiny walls (C) the ultrasonic sensor cannot measure differences between the outgoing and the incoming signal.

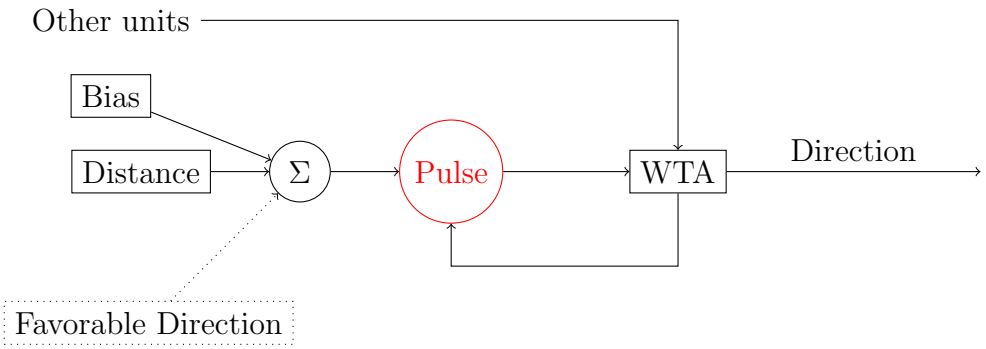

Figure 8.1: A neural network to calculate the most favorable direction for the robot

Also in this case a bottom threshold has to be implemented and the robot relies on the visual sensor.

Chapter 9

Conclusion

A classic CSP is made up by three components: the involved variables, their domain containing the possible values and the constraints. The goal is to assign a value to each variable such that each constraint is fulfilled. This solution is called a complete solution. An important definition in terms of CSP is the energy function. Each violation of a constraint increases the energy of the current state. Therefore, a global state which is a complete solution is found if the energy function is 0. A graphical representation of the relation is given by the constraint graph.
A daily example is the scheduling of lectures. This is a feature of CSPs: they occur in daily life. Therefore, a lot of research is done in this area. The clear structure enables the quick understanding of the problem. However, the problem can become complex very quickly as CSP is known to be NP-complete. With the rising complexity the time to solve the problem rises as well. The classical algorithms can even get stuck during the process. Another problem is that higher-order constraints, i.e. constraints which involve more than two variables, are hard to solve and have to be broken down into lower orders.
The classical approaches work sequentially as they change or assign one value at a certain time. The algorithms check if an assigned value has an energy greater than 0 and try to find another value of the domain to decrease the energy value. For CSPs – known to be NP-complete – this takes exponential time. Heuristics have been invented to help the algorithm find a promising variable to start or proceed with. The algorithms are not able to learn. When solving complex CSPs classically parallelization is crucial to be able to find a solution in an acceptable time span. With the powerful processors on the market the parallelization can be done efficiently. Hence, the time to solution can be reduced significantly. It is of great importance to invest more in research on this topic to improve the existing methods and come up with fast implementations as the algorithms are actually mature.
Another object of research is the implementation of neural networks as problem solver which has proven to be very efficient compared to classical approaches. A network consists of neural units which compute the solution simultaneously depending on the state of the connected units. They are linked to each other by weighted

connections. The input is processed by a dedicated activation function which is different among the models. In some models the connection weights even adapt during a learning process depending on the excitation rate of the connected units. Therefore, the system can come up with a solution in a shorter time span and will avoid making mistakes.
This is why neural networks are highly attractive in terms of CSP. Nevertheless, due to the binary connections between the units only unary and binary constraints can be implemented. Higher-order constraints have to be modeled with intermediate neurons leading to a high number of implemented neurons. In local, fixed environment neural networks have shown to be a successful method. However, an implementation in a "real" environment is required to prove their maturity.
In the robotics sector several challenges come up like path planning or exploration of a network. These robotic tasks can be formalized as CSPs. As humans face CSPs every day the treatment with its clear structure can be understood quickly by the developers. Therefore, accurate models of the problem can be derived with ease. Basically, the model for a robotics task consists of the position of the robot which is defined as the variable of the CSP. Its possible future directions are the values of the domain. The constraints state that the robots must not crash into the walls and into each other. Additionally, each task generates its own constraints. During the process of CSP solving the system updates the set of most favorable directions depending on the constraints and the current position. The direction which violates the least constraints is chosen. Certainly, neural networks can be implemented as problem solvers as shown in the case study. The implementation proves to be efficient as the network calculates the values for the variables simultaneously and is capable of learning the relations between the constraints and the environment. Furthermore, the WTA circuit enables the network to decide rather quick on the most favorable direction as the winning value of the domain inhibits the other cells via the plastic connectors.
The calculation can also be combined with advantages of classical approaches like distributed CSP. In a DCSP the agents calculate the basic information for themselves and share only the results with the others. This is also interesting for the application as the robots can interact as individuals which is a main aspect of human behavior. The more realistic the robots' behavior is the more mature the solution for the task. The computing power can be split and only a smaller amount of data has to be processed by a centralized unit where appropriate. Nevertheless, the number of robots involved can hinder themselves to find a good solution. Therefore, the existing algorithms have to developed even further to eliminate this problem.

List of Figures

Bibliography

[BCM82] Elie L. Bienenstock, Leon N. Cooper, and Paul W. Munro. Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *The Journal of Neuroscience*, 2(1):32–48, 1982.

[BPS99] Sally C. Brailsford, Chris N. Potts, and Barbara M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999.

[CB08] Leon N. Cooper and Brian S. Blais. Bcm theory. 3(3):1570, 2008. revision 91041. URL: `http://www.scholarpedia.org/article/BCM_theory` [cited 14.06.2014].

[CCDA11] Yves Caniou, Philippe Codognet, Daniel Diaz, and Salvador Abreu. Experiments in Parallel Constraint-Based Local Search. In Merz, P and Hao, JK, editor, *Evolutionary Computation In Combinatorial Optimization*, volume 6622 of *Lecture Notes in Computer Science*, pages 96–107, 2011.

[DBD+09] Arnaud Doniec, Noury Bouraqadi, Michael Defoort, Van Tuan Le, and Serge Stinckwich. Distributed constraint reasoning applied to multi-robot exploration. In *Ictai: 2009 21St International Conference On Tools With Artificial Intelligence*, Proceedings-International Conference on Tools With Artificial Intelligence, pages 159–166, 2009.

[Heb59] Donald O. Hebb. *The Organization of Behavior A Neuropsychological Theory*. John Wiley and Sons, Inc., 5 edition, 1959.

[Hor06] Timothy K. Horiuchi. A neural model for sonar-based navigation in obstacle fields. In *Proceedings of The 2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. ISCAS, May 2006.

[HSA84] Geoffrey E. Hinton, Terrence J. Sejnowski, and David H. Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Technical report, 1984.

[KD94] Simon Kasif and Arthur L. Delcher. Local consistency in parallel constraint-satisfaction networks. *Artificial Intelligence*, 69:307–327, 1994.

[MMH13] Raziyeh Moghaddas, Eghbal Mansoori, and Ali Hamzeh. A new multi-agent algorithm for solving constraint satisfaction problems. In *Information and Knowledge Technology (IKT), 2013 5th Conference on*, pages 197–201, 2013.

[MMI13] Hesham Mostafa, Lorenz. K Mueller, and Giacomo Indiveri. Recurrent networks of coupled winner-take-all oscillators for solving constraint satisfaction problems. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 719–727. Curran Associates, Inc., 2013.

[NA00] Nicoleta Neagu and Whitestein Technologies AG. *Constraint Satisfaction Techniques for Agent-based Reasoning*. Birkhäuser Verlag, Basel, 2000.

[NF01] Nicoleta Neagu and Boi Faltings. Exploiting interchangeabilities for case adaptation. In Aha, DW and Watson, I, editor, *Case-Based Reasoning Research And Development, Proceedings*, volume 2080 of *Lecture Notes In Artificial Intelligence*, pages 422–436, 2001.

[RN02] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 5. Prentice Hall, 2 edition, 2002.

[RRM13] Jean-Charles Regin, Mohamed Rezgui, and Arnaud Malapert. Embarrassingly Parallel Search. In Schulte, C, editor, *Principles And Practice Of Constraint Programming, Cp 2013*, volume 8124 of *Lecture Notes in Computer Science*, pages 596–610, 2013.

[Rya10] Malcolm Ryan. Constraint-based multi-robot path planning. In Rakotondrabe, M and Ivan, IA, editor, *2010 Ieee International Conference On Robotics And Automation (Icra)*, IEEE International Conference on Robotics and Automation ICRA, pages 922–928. IEEE, 2010.

[SAS90] P.S. SASTRY. Stochastic Networks for Constraint Satisfaction and Optimization. *Sadhana-Academy Proceedings in Engineering Sciences*, 15(4-5):251–262, 1990.

[SN93] Gerhard Seiler and Josef A. Nossek. Winner-Take-All Cellular Neural Networks. *Ieee Transactions On Circuits And Systems Ii-Analog And Digital Signal Processing*, 40(3):184–190, 1993.

[ST02] N Sadati and J Taheri. Solving robot motion planning problem using Hopfield Neural Network in a fuzzified environment. In *Proceedings Of*

The 2002 Ieee International Conference On Fuzzy Systems, Vol 1 & 2, pages 1144–1149. IEEE; IEEE Neural Network Soc, 2002.

[WC12] David Weikersdorf, Jörg Conradt. Event-based Particle Filtering for Robot Self-Localization, Proceedings of the IEEE International Conference on Robotics and Biomimetics (IEEE-ROBIO), pages: 866-870, Guangzhou, China, 2012.

[WH13] David Weikersdorfer, Raoul Hoffmann,and Jorg Conradt. Simultaneous Localization and Mapping for event-based Vision Systems, International Conference on Computer Vision Systems (ICVS), p. 133-142, St. Petersburg, Russia, 2013.

[YDIK98] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formal-ization and algorithms. Ieee Transactions On Knowledge And Data En-gineering, 10(5):673–685, 1998.

[YW00] Shengxiang Yang and Dingwei Wang. Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. Ieee Transactions On Neural Networks, 11(2):474–486, 2000.

License