LU TP 00-40

An Information-Based Neural Approach to Generic Constraint Satisfaction

Henrik Jönsson[1] and Bo Söderberg[2]

Complex Systems Division, Department of Theoretical Physics
Lund University, Sölvegatan 14A, S-223 62 Lund, Sweden
http://www.thep.lu.se/complex/

To appear in *Artificial Intelligence* **142:1** 1–17 (Nov. 2002).

Keywords: constraint satisfaction, graph coloring, connectionist, artificial neural network, mean-field annealing, heuristic, information.

Abstract

A novel artificial neural network heuristic (INN) for general constraint satisfaction problems is presented, extending a recently suggested method restricted to boolean variables. In contrast to conventional ANN methods, it employs a particular type of non-polynomial cost function, based on the information balance between variables and constraints in a mean-field setting. Implemented as an annealing algorithm, the method is numerically explored on a testbed of Graph Coloring problems. The performance is comparable to that of dedicated heuristics, and clearly superior to that of conventional mean-field annealing.

[1] henrik@thep.lu.se
[2] Bo.Soderberg@thep.lu.se

1 Introduction

Artificial Neural Networks (**ANN**) have provided a versatile heuristic approach to combinatorial optimization and constraint satisfaction [10, 19, 7, 1, 5]. In a conventional ANN approach, a constraint satisfaction problem (**CSP**) is attacked by converting it to an optimization problem, attempting to minimize a non-negative cost function vanishing only for solutions.

In a recent paper [13] an improved ANN approach (**INN**), directly aimed at boolean CSP, was presented. Based on information-theoretical considerations, it utilized a very specific, strongly non-linear cost function. Numerical explorations on a testbed of K-SAT instances yielded a performance substantially better than that of a conventional ANN approach, and comparable to that of a dedicated heuristic – gsat+walk [20]. This improvement can be understood, at least partly, as being due to a progressively increased sensitivity to the breaking of constraints, stemming from the nonlinearity of the INN cost function.

In this paper we provide an extension of the boolean INN approach to more general constraint satisfaction by utilizing an encoding in terms of Potts spins, and present an annealing algorithm based on this approach. Like in the boolean case, it is derived from an analysis of the information balance between mean-field variables and constraints, in a mean-field (**MF**) approximation. The result is a general-purpose CSP heuristic.

As a specific application example we have chosen *Graph Coloring* (**GC**) [17], and we provide a detailed discussion of the implementation of INN for this problem type.

The method is numerically explored for GC with three colors on a large testbed of random graphs with varying edge densities. Also the performance on a small set of more dense graphs, available from DIMACS[3], are investigated. Like for the boolean K-SAT problems, a substantially improved performance is observed, as compared to a conventional ANN approach. To gauge the performance, we have applied two dedicated heuristics to the testbed; a biased simulated annealing approach, *SAU* [3, 11], and a simple search heuristic, *DSATUR* [2, 4]. A well known SAT solver, *gsat+walk* [20], is also used for comparison.

The structure of the paper is as follows: In Section 2, we present a general derivation of the method for a generic CSP, based on information analysis. In Section 3, we describe the specific application of the method to GC, and discuss algorithmic details. Section 4 contains a description of the numerical explorations and the testbeds, as well as a presentation and a discussion of the results. Finally, Section 5 contains a brief summary and our conclusions.

[3] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf/`

2 INN for Non-boolean (Potts) Systems - General Derivation

In this section we will, in the context of general CSP, explain the ideas behind the INN method, which are based on analyzing the information content in the constraints to be satisfied.

2.1 General CSP: notation

In a CSP instance a set X of N discrete variables, $X = \{x_1 \dots x_N\}$, is considered. We assume the domain $\mathcal{D}_i = \mathcal{D}$ of each variable x_i to have C possible states, $\mathcal{D} = \{1 \dots C\}$. Further, a set of M constraints,

$$f_m(X) = 0, \quad m = 1 \dots M, \tag{1}$$

is given; each constraint is assumed to involve a subset of size K of the N variables. The constraints define a (possibly empty) subset $\mathcal{S}$ of the state space $\mathcal{D}^N$ consisting of the solutions: $\mathcal{S} = \{X \in \mathcal{D}^N | f_1(X) = \dots = f_M(X) = 0\}$.

The question is whether such a CSP instance is solvable, i.e. whether $\mathcal{S}$ is non-empty. Finding a solution suffices to prove solvability; proving non-solvability is in general harder.

We limit the discussion to fixed C and K for simplicity only – the method can easily be adapted to problem types with varying C as well as K.

2.2 Conventional MF ANN Approach

For ANN and INN we will adopt a slightly different encoding, and map the variables onto $\mathcal{R}^C$ by associating with each variable x_i a *Potts spin* [21], a C-dimensional vector $\mathbf{s}_i = (s_{i1}, s_{i2}, \dots, s_{iC})$, such that the assignment $x_i = c$ is represented by $\mathbf{s}_i = \mathbf{e}_c$, where $\mathbf{e}_c$ is the principal vector in the c-direction, defined to have a unit cth component, while the other components vanish.[4] Defining $\mathbf{e}$ as the uniform C-dimensional vector $(1, 1, \dots 1)$, the domain of each Potts spin is given by $\hat{\mathcal{D}} = \{\mathbf{s}_i \in \{0, 1\}^C | \mathbf{e} \cdot \mathbf{s}_i = 1\}$.

Obviously, every function $f(X)$ of the original variables can be written as an equivalent function $F(\mathbf{s})$ of the set of spins $\mathbf{s} = \{\mathbf{s}_1 \dots \mathbf{s}_N\}$. As an example, an arbitrary real unary function $f(x_i)$ is mapped to the linear function $F(\mathbf{s}_i) = \mathbf{f} \cdot \mathbf{s}_i$, where $\mathbf{f} = (f(1), \dots, f(C))$.

[4] This in order to enable the MF approximation, requiring variables to reside in a linear space.

In the conventional ANN approach, a CSP instance is transformed into an optimization problem, encoded in terms of a non-negative cost function E over the state space $\hat{\mathcal{D}}^N$, such that $E(\mathbf{s})$ vanishes iff $\mathbf{s}$ corresponds to a solution. E is typically chosen as a polynomial in $\mathbf{s}$, and written such that each term is the product of components from distinct Potts spins (*multilinearity*). The mth constraint is defined above (equation (1)) as the vanishing of a distinct function $f_m(X)$, depending on a subset $\{x_i | i \in \Omega_m\}$ of size K of the variables. Then, a suitable associated cost, E_m, can be defined as a polynomial of degree K in the associated spins $\{\mathbf{s}_i, i \in \Omega_m\}$. Specifically, E_m can be chosen as

$$E_m(\mathbf{s}) = \sum_{c_1,\ldots,c_K} s_{i_1,c_1} \ldots s_{i_K,c_K} \Theta_{c_1 \ldots c_K}, \tag{2}$$

where Θ is zero if the value combination $(c_1, \ldots, c_K)$ for the K variables $(x_{i_1}, \ldots, x_{i_K})$ involved solves the constraint, and unity otherwise. Then the total cost function $E(\mathbf{s})$ is defined by summing the contributions from the individual constraints, i.e. $E(\mathbf{s}) = \sum_{m=1}^{M} E_m(\mathbf{s})$.

2.2.1 The Mean Field Approximation

A simple ANN heuristic is given by minimizing the cost function with respect to one Potts neuron at a time when searching for the global minimum. The derivative $\partial E / \partial s_{ic}$ yields a measure of the number of constraints broken if the variable x_i would be assigned the value c. A problem with such an algorithm is that the state very easily gets stuck in a suboptimal local minimum.

One way to avoid suboptimal local minima is to use a simulated annealing scheme [14], where the system is placed in a virtual heat bath represented by the simulation of a Boltzmann distribution over the state space $\hat{\mathcal{D}}^N$, such that the probability of a state $\mathbf{s}$ is proportional to $\exp(-E(\mathbf{s})/T)$, where T is an artificial temperature to be slowly lowered. At high T all states are about equally probable, while as $T \to 0$, the support of the distribution shrinks to contain only the states with the lowest cost (the solutions for a solvable problem), all equally probable.

In an ANN annealing approach one instead adopts the mean field (**MF**) approximation [18], where the thermal averages $\langle \mathbf{s}_i \rangle$ of the spins $\mathbf{s}_i$ are approximated by estimates $\mathbf{v}_i$ (*MF Potts neurons*), obeying a self-consistent set of equations, the *MF equations*, given by

$$v_{ic} = \frac{\exp(u_{ic})}{\sum_d \exp(u_{id})}, \tag{3}$$

$$u_{ic} = -\frac{1}{T} \partial E(\mathbf{v}) / \partial v_{ic}. \tag{4}$$

where the cost function E appears with argument $\mathbf{v}$ rather than $\mathbf{s}$, representing the thermal average $\langle E(\mathbf{s})\rangle$.

Note that a MF neuron $\mathbf{v}_i$ is constrained to the convex hull in $\mathcal{R}^C$ of the domain $\hat{\mathcal{D}}$ of the corresponding Potts spin $\mathbf{s}_i$, and can be seen as fuzzy version of $\mathbf{s}_i$. In particular, $v_{ic} \geq 0$ and $\sum_c v_{ic} = 1$, and v_{ic} can be interpreted as a probability of the assignment $x_i = c$.

2.2.2 ANN MF Annealing

In conventional MF annealing [19, 7], the MF equations (3,4) are solved iteratively (thus defining a deterministic dynamics), updating one MF neuron at a time, while the temperature is slowly decreased from an initial high value where roughly uniform neurons result, $v_{ic} \approx 1/C$.

As $T \to 0$, the neurons are gradually pushed "on shell", with components approaching 0 or 1, representing a set of definite assignments - the output of the algorithm; if this defines a solution, the CSP instance is proven solvable. A typical MF annealing algorithm is presented in figure 1.

1. Set the temperature,T, to a high value and initialize $\mathbf{v}$ close to the high-T fixed point, $v_{ic} = 1/C$, with small random deviations (a few %).
2. For each node i in turn:
   (a) Compute $\mathbf{u}_i$ using equation (4).
   (b) Update $\mathbf{v}_i$ according to equation (3).
3. If any of the stop criteria is met, go to 5.
4. Lower T by a fixed annealing factor, and go to 2.
5. Extract a candidate solution by for each node i choosing a color corresponding to the largest component of $\mathbf{v}_i$.

A typical stop criteria is if the *saturation* is close to one; $\frac{1}{N}\sum_{ic} v_{ic}^2 > 1 - \epsilon$, where ϵ is a small number.

Figure 1: The MF annealing algorithm.

2.2.3 Variational MF Derivation

The MF approximation can be derived from a variational principle, where the true Boltzmann distribution $\propto \exp(-E(\mathbf{s})/T)$ is approximated by one with independent probabilities for each variable, defined by the components of $\mathbf{v}_i$. These are optimized by minimizing a free energy, given by

$$F(\mathbf{v}) = T \sum_{ic} v_{ic} \log(v_{ic}) + E(\mathbf{v}), \tag{5}$$

with the restriction that $v_{ic} \geq 0$ and $\sum_c v_{ic} = 1$. The first term amounts to $-TS(\mathbf{v})$, where S is the entropy associated with $\mathbf{v}$. At an extremal value we must have $\partial F/\partial v_{ic} = \lambda_i$, where λ_i is a Lagrange multiplier for the unit-sum constraint on $\mathbf{v}_i$. This yields

$$T \log(v_{ic}) + T + \partial E(\mathbf{v})/\partial v_{ic} = \lambda_i, \tag{6}$$

which leads to the MF equations (3,4).

2.3 INN Approach

We will now define INN by means of an information-based modification of ANN, where the polynomial cost function $E(\mathbf{v})$ is replaced by a specific non-polynomial one, based on an information-theoretical analysis of the constraints. For each constraint this typically leads to the negative logarithm of a polynomial function yielding unity if the constraint is satisfied, and zero if broken. This yields a divergent cost for a non-solution, which in a MF setting leads to a sensitivity to softly broken constraints that progressively increases with the severity.

2.3.1 INN cost function

To construct such a cost function, we rely on the approximating assumption (inherent in the MF approximation) that the Boltzmann distribution at each temperature factorizes into a product of single-variable distributions,

$$P(X) = \prod_i p_i(x_i). \tag{7}$$

Each single-variable distribution p_i is completely defined by an MF neuron $\mathbf{v}_i$, with the probability for $x_i = c$ given by v_{ic}. Thus, $\mathbf{v}$ can be seen as a parameterization of P.

Now, assume the mth constraint to be defined by a function f_m of a subset of the variables as in equation (1). Then, the probability distribution P, as parameterized by

$\mathbf{v}$, gives a well-defined probability that the constraint be unsatisfied, given by $\langle E_m(\mathbf{s})\rangle$, which reduces to $E_m(\mathbf{v})$ with the multilinear E_m of equation (2).

The amount of information required to force a constraint to be satisfied can be estimated as $-\log(1 - E_m(\mathbf{v}))$, and as a cost function we take the sum of this over the set of constraints, i.e.

$$I(\mathbf{v}) = \sum_m -\log(1 - E_m(\mathbf{v})), \tag{8}$$

which is thus an estimate of the total amount of information required for solving the problem.

As a comparison, the polynomial ANN cost function $E(\mathbf{v}) = \sum_m E_m(\mathbf{v})$, corresponds to a Boltzmann distribution over the state space given by $P_E(\mathbf{s}) \propto \prod_m \exp(-E_m(\mathbf{s})/T)$ with a penalty factor of $\exp(-1/T)$ for each broken constraint.

The INN cost function I formally corresponds to the more radical Boltzmann distribution

$$P_I(\mathbf{s}) \propto \prod_m (1 - E_m(\mathbf{s}))^{\frac{1}{T}}\,, \tag{9}$$

which, since $1 - E_m(\mathbf{s})$ yields 1 for a solution and 0 for a non-solution, vanishes for all non-solutions and yields a uniform non-zero weight for all solutions, independently of T (the T dependence appears only in the MF equations).

2.3.2 Variational Interpretation

Formally, the modified MF approximation employed in INN can be seen as the result of the minimization of a variational free energy,

$$F(\mathbf{v}) = -TS(\mathbf{v}) + I(\mathbf{v}), \tag{10}$$

where the first term can be interpreted as a measure of the unused information resources associated with the MF neurons. At high T, all states are about equally probable, and $v_{ic} \approx 1/C$. The information content $-S$ is then high, amounting to $\log(C)$ per neuron (one bit for $C = 2$). As T is lowered, a specific state is chosen for each variable, and the information resources are used up, $-S \to 0$.

Thus, in the INN approach, the variables can be seen as information *resources*, that are gradually used to satisfy the constraints, seen as information *consumers*. [5] At high T (typically above a well-defined critical temperature, T_c), the resources are intact, but as T is decreased, an increasing pressure is applied towards spending them.

[5] This might seem as an abuse of the information concept; an alternative is to view the spins as resources for *storage* of the information produced by the constraints.

2.3.3 INN MF Annealing

When using the INN cost function in place of the conventional ANN one in the MF equations (3,4), a slight modification of the iterative dynamics is needed due to the non-polynomial nature of the former, in order to avoid instabilities [16, 13]. Equation (4), which is relevant for a multilinear polynomial cost function, is replaced by

$$u_{ic} = -\frac{I(\mathbf{v})|_{\mathbf{v}_i=\mathbf{e}_c}}{T}, \tag{11}$$

where $\mathbf{e}_c$ as before represents the sharp c-state, with a unit cth component and the rest zero. This modification amounts to using cost *differences*, rather than derivatives as in equation (4); for a multilinear cost function the results coincide.

Due to the singular behavior of the logarithm in equation (8) and thus in equation (11) for small arguments, it may happen that one or more components of $\mathbf{u}_i$ become divergent ($-\infty$) within the numerical resolution, which is the case when a constraint is close to becoming fully broken for a particular choice of state $\mathbf{e}_c$, which typically happens at a low T.

If not all choices yield a divergence, there is no problem: The corresponding components of $\mathbf{v}_i$ will become zero. However, in cases where a divergence is inevitable, a regularization method has to be devised.

To that end, a counter n_{ic} is assigned to each state c of the variable x_i, initialized to zero, and incremented for each constraint that would make a singular contribution (which is disregarded) to u_{ic}. In cases where all the counters for a fixed i are non-zero, only the states c with the lowest count are considered, and one possibility (*deterministic* method) is to set the corresponding components of $\mathbf{v}_i$ to equal values summing up to one, while the others are set to zero, thus ignoring the finite contributions to $\mathbf{u}_i$. In [13] an alternative, *stochastic* method was used, where a random state c is selected among those with the lowest count, and the corresponding component of $\mathbf{v}_i$ set to unity, the rest to zero. This version leads to a stochastic search in the low temperature region, resulting in an improvement in performance, but an increase in time consumption.

In the low temperature limit where all non-zero contributions are infinite, the stochastic regularization method yields a dynamics resembling a class of non-annealing ANN algorithms used on constraint satisfaction problems [1, 5], while the deterministic method yields a low T behavior closer to that of conventional MF annealing.

In practice, when evaluating the INN cost function in equation (11) it is faster to take the logarithm of a product than to sum the logarithms. It may happen that the product vanishes numerically, yielding a divergent logarithm; this is treated as an extra divergent contribution, and the corresponding counter is incremented.

With these modifications, INN annealing proceeds just like conventional ANN annealing: The modified MF equations are solved iteratively in a serial manner, with annealing in T, etc. However, due to the nonlinearity of the logarithm in the INN cost function, equation (8), constraints on their way to becoming unsatisfied are detected on an earlier stage, leading to a better revision capability for INN as compared to ANN, and a substantially improved performance.

3 Application to Graph Coloring

Now we leave the general discussion in favor of an application to a specific problem type, for which we have chosen *Graph Coloring* (**GC**) [17].

In GC, a graph of N nodes and M edges is given, and C distinct colors. Each node is to be assigned a color, such that each edge connects nodes of distinct colors. Thus, there is one constraint for each edge, involving $K = 2$ variables.

3.1 INN for Graph Coloring

We assign a Potts spin $\mathbf{s}_i$ to encode the coloring of each node i, $i = 1, \ldots, N$, and employ the (modified) MF approximation, yielding MF neurons $\mathbf{v}_i$.

The scalar product $\mathbf{s}_i \cdot \mathbf{s}_j$ yields unity if the nodes i, j are in the same state, and zero if not. The analogous expression with MF neurons, $\mathbf{v}_i \cdot \mathbf{v}_j$, measures the probability that the nodes are in the same state, which is precisely what we need for E_m, so let

$$E_m(\mathbf{v}) = \mathbf{v}_{i_m} \cdot \mathbf{v}_{j_m}, \tag{12}$$

where i_m, j_m label the two nodes connected by the edge m.

Let $\mathbf{J}$ denote the connection matrix for the graph, i.e. $J_{ij} = 1$ if nodes i, j are connected, zero otherwise. Then the INN cost function $I = -\sum_m \log(1 - E_m)$ can be expressed as

$$I = -\frac{1}{2} \sum_{i,j} J_{ij} \log\left(1 - \mathbf{v}_i \cdot \mathbf{v}_j\right). \tag{13}$$

The INN MF equations become $v_{ic} = exp(u_{ic}) / \sum_d \exp(u_{id})$, with

$$u_{ic} = \frac{1}{T} \sum_j J_{ij} \log\left(1 - v_{jc}\right), \tag{14}$$

where care must be taken to regularize singular contributions, as discussed in Section 2.3.3. Note that a component of $\mathbf{u}_i$ gets a singular contribution from an edge only in the limit of the corresponding component of $\mathbf{v}_j$ being unity (within the numerical resolution), which is more likely to happen at low T.

3.1.1 Critical T Discussion

Due to the symmetry with respect to the permutation of colors, the MF equations will always have a symmetric solution

$$v_{ic} = \frac{1}{C}, \tag{15}$$

defining a trivial fixed point of the dynamics. By means of a linearization around this fixed point, the dynamics for infinitesimal deviations $\epsilon_{ic} = v_{ic} - 1/C$ becomes

$$\epsilon_{ic} = -\frac{1}{T(C-1)} \sum_j J_{ij} \left(\epsilon_{jc} - \frac{1}{C} \sum_d \epsilon_{jd} \right). \tag{16}$$

A simple analysis yields that the trivial fixed point is stable for high enough T, but suffers a destabilizing bifurcation at a critical temperature T_c, given by $-\frac{\lambda}{C-1}$ where λ is the largest (in absolute value) negative eigenvalue of $\mathbf{J}$. T_c serves as a suitable initial temperature in the annealing algorithm, and can easily be estimated. For a graph with at least one edge, $1/(C-1)$ is a strict lower bound for T_c.

4 Numerical Explorations

2-coloring ($C = 2$) is known to be of polynomial complexity, and we will focus on 3-coloring ($C = 3$), which is NP-complete [17].

Problem difficulty depends on the edge density $\gamma = 2M/N$. For 3-coloring, there exists a critical edge density $\gamma_c \approx 4.6$ [9], such that in the large-N limit all problems are solvable below γ_c, and unsolvable above. (Such phase transitions are known to exist in many classes of CSP [8, 15].)

Although the problem of finding a solution becomes increasingly difficult with increasing γ, the decidability problem is most difficult around γ_c; at lower γ a solution is easy to find, while at higher γ unsolvability is more likely to be easily provable.

In addition to 3-coloring we have also applied the algorithms to a limited set of problems on more dense graphs, widely studied in the literature (see e.g. [12, 11, 5]). The problem

instances studied are labeled g125-17, g125-18, g250-15 and g250-29, where the first two actually correspond to the same 125-node graph, to be colored with respectively 17 and 18 colors.

4.1 3-Coloring Testbeds

To probe the performance of INN annealing as applied to 3-coloring, we have chosen a testbed of random graphs, with edge densities in the neighborhood of γ_c. To be specific, we have used γ values between 3.4 and 4.6 in steps of 0.2, and in addition 4.1 and 4.3. Problem sizes probed are $N = 250, 500, 1000, 2000$. For a given edge density and problem size, the edge count is given as $M = \gamma N/2$, and a set of 200 random problem instances is generated by, for each instance, choosing at random M of the $N(N-1)/2$ possible edges.

4.2 Preprocessing

A simple preprocessing is made on the graphs before they are handed to the algorithms. Nodes with less than $C = 3$ neighbors are removed from the graph, since they can be trivially colored. This is done recursively until the remaining nodes have at least C neighbors. In table 1 the average sizes of the graphs after preprocessing is shown for some original values of N and γ. The preprocessing can be expected to improve

	$\gamma = 3.6$			$\gamma = 4.2$			$\gamma = 4.6$		
N	M	N'	M'	M	N'	M'	M	N'	M'
250	450	130	259	525	181	414	575	201	496
500	900	253	506	1050	361	827	1150	401	990
1000	1800	509	1020	2100	720	1649	2300	799	1973
2000	3600	1025	2055	4200	1437	3293	4600	1596	3943

Table 1: Average problem size reduction due to preprocessing. N, M and γ are the original problem parameters, while N' and M' denote the reduced node and edge counts.

speed about equally much for the different algorithms, which is confirmed by empirical test runs. These also indicate a comparable performance improvement in the form of a small change (slightly larger for ANN) in the position in γ of the (algorithm-dependent) apparent phase transition.

4.3 Algorithmic Details for INN

The temperature is initially set close to an estimate of the critical temperature $T_c = -\frac{\lambda}{1-C}$, where λ is obtained by iterating $\mathbf{x} \rightarrow (const \times \mathbf{1} - \mathbf{J})\,\mathbf{x}$ a few dozen times, with a suitable random initial vector $\mathbf{x}$. A schematic description of the algorithm is given in figure 2. The deterministic version (as discussed in Section 2.3.3) is used for

1. Set $T \approx T_c$, and initialize $\mathbf{v}$ close to the high-T fixed point, $v_{ic} = 1/C$, with small random deviations (a few %).
2. For each node i in turn:
   (a) Compute $\mathbf{u}_i$ using equation (11).
   (b) Update $\mathbf{v}_i$ according to equation (3). (In case of singular components in $\mathbf{u}_i$, this pair of steps is modified as discussed in Section 2.3.3).
3. If any of the stop criteria is met, go to 5.
4. Lower T by a fixed annealing factor, and go to 2.
5. Extract a candidate solution by for each node i choosing a color corresponding to the largest component of $\mathbf{v}_i$.

Figure 2: The INN annealing algorithm for GC.

3-coloring. For the annealing factor we have experimented with a few different values. Slower annealing tends to improve the quality for large and difficult problems, but at the cost of more CPU time used. There is also a limit where slower annealing does not improve the quality anymore, and a restart of the algorithm gives a better payoff. The presented results are consistently based on an annealing rate of 0.99, and we have allowed for a maximum of ten updates of all neurons per temperature, to allow the state to converge ($\max_{ic} |\Delta v_{ic}| < 0.1$).

At every tenth temperature we apply two stop criteria. First, a temporary sharp configuration $\mathbf{s}$ is extracted from $\mathbf{v}$, and if $\mathbf{s}$ is a solution the algorithm stops. The second criterion applies if the neurons are saturated ($\sum_i \sum_c v_{ic}^2 > 0.9N$) and stable ($\max_{ic} |\Delta v_{ic}| < 0.01$). In addition, if no solution has been found until $T < 0.3$ the algorithm aborts.

4.4 Comparison Algorithms

The performance of INN annealing has been compared to that of four other algorithms: **1)** conventional ANN annealing, **2)** a biased simulated annealing algorithm, SAU [3, 11], **3)** a heuristic, DSATUR [2, 4], and **4)** a SAT heuristic, gsat+walk[20], all applied to the same testbed. Where not otherwise stated, the algorithms are implemented by the authors.

Comparing different algorithms is quite difficult, as they have different time scales at which they are most efficient. Also, each algorithm has different optimal parameter settings for different N and γ. Despite this, we have for each algorithm for simplicity used the same parameter settings on the entire testbed.

The parameters are chosen by means of testruns on a small distinct testset of instances, drawn from the same ensemble as the production testbed. We have chosen a set of suitable parameters (annealing parameter, number of updates per temperature) for the INN algorithm first, and then attempted to optimize the parameter settings for the other algorithms, given that they are allowed to use about the same time as INN. Note that our testset contains solvable as well as unsolvable problems. Hence parameters are chosen such that the algorithms are completed within a given maximal time, also for unsolvable problems.

4.4.1 Conventional ANN annealing

A suitable ANN cost function based on equation (12) is

$$E(\mathbf{v}) = \frac{1}{2} \sum_{i,j} J_{ij} \mathbf{v}_i \cdot \mathbf{v}_j, \tag{17}$$

yielding the MF equations (3) with

$$u_{ic} = -\frac{1}{T} \sum_{j} J_{ij} v_{jc}. \tag{18}$$

T is initialized close to the critical temperature, given by $(C-1)/C$ times that for INN. We have used an annealing rate of 0.99, and the same stop criteria as in INN, except for the stop temperature, which is set to 0.1 instead of 0.3 (lower than for INN, where the non-linear cost function makes the neurons saturate faster).

4.4.2 SAU

A number of simulated annealing (SA) [14] approaches have been devised for graph coloring problems [11, 3]. For 3-coloring it is appropriate to use one with a fixed number of colors where the goal is to minimize the number of broken edges. In [11], where a set of SA algorithms were tested with graphs with varying edge densities, this was also shown to be the best strategy for graphs with a low edge density. In such an approach each node in the graph is assigned a variable, $x_i = 1, .., C$ representing the color of the node. A cost analogous to equation 17, defined as

$$E = \frac{1}{2} \sum_{i,j} J_{ij} \delta_{x_i x_j}, \tag{19}$$

can be used. Local moves are selected by choosing a node i and a random change in value for x_i. To guide the search into low cost states a virtual temperature parameter, T, is introduced and moves are accepted with a probability $p = \min\left(\exp\left(-\Delta E/T\right), 1\right)$, where $\Delta E = E_{new} - E_{old}$. If the temperature is high all moves are accepted, while at low temperature uphill moves (increased cost) are rejected.

In [11, 3], an algorithm of this type is described, which we will refer to as SAU. There, a restricted move class is used, where only nodes contributing to the cost (as opposed to all nodes) are allowed as candidates for a color change; this is empirically more efficient. A candidate move is given by 1) choosing a random broken constraint, 2) picking a randomly chosen variable in the constraint, and 3) changing the value into a random (different) color.

The move class is not symmetric, and this algorithm corresponds to a kind of biased simulated annealing, which does not necessarily yield an emulation of a Boltzmann distribution. Also, ergodicity seems to be broken: All possible states cannot be reached from any initial state; this does not have to be a disadvantage.

The algorithm starts in a random state and at a high temperature to allow for uphill moves. A certain number ($2N$) of attempted moves (accepted or not) define an iteration; between iterations the temperature is decreased by a fixed factor (0.97). This is repeated until a solution is found, or the cost has not changed over a certain number (10) of iterations. To optimize the algorithm (within the time used by INN) we have tested different annealing factors and different numbers of attempted moves at each temperature.

4.4.3 DSATUR

The DSATUR algorithm is designed to answer the question how many colors are needed to color a graph; it always succeeds, and returns the number of colors used. If this is

less than or equal to C, a solution to C-coloring is implied.

DSATUR starts with all nodes uncolored, and selects nodes to color one by one. The order in which nodes are selected is dynamically determined by the number of colors that cannot be used because of already colored neighbor nodes. In each step the node with the smallest number of available colors is selected to be colored; if several, a random selection is made.

This algorithm was presented by D. Brélaz [2], and we have used a program made by J. Culberson [4] available from the *world-wide web* [6]. In [4] a set of similar algorithms was presented, with varying rules for ordering the nodes. Our choice of DSATUR among these was based on preliminary experiments on the class of problems used in our testbed, indicating that DSATUR performed best. With the random choice of node at a degeneracy there is no parameter to optimize, and the algorithm is restarted until a solution is found or the time limit is reached.

4.4.4 Gsat+walk

Gsat+walk is a heuristic designed for satisfiability (*SAT*) problems, but it can be used also for graph coloring problems, as any GC instance can be transformed into a SAT instance in polynomial time.

In a SAT problem [17, 6] there are N boolean variables, x_i =True/False, $i = 1, ..., N$. In conjugate normal form the constraints are given by clauses, $(a_1 \vee a_2 \vee ... \vee a_K)$, where each of the K literals, a_j, is given by a variable x_{i_j} or its negation $\neg x_{i_j}$, and $\vee$ represents the inclusive or. Then it is enough that one variable is in the correct state for the clause to be satisfied. A problem instance is given by M clauses and the question is whether there is a configuration of the variables that satisfies all clauses.

The transformation from a graph coloring instance to a SAT instance can be done using boolean variables x_{ic} defined as True if node i is assigned color c and False otherwise. Each edge i, j in the graph then yields a clause for each color $c = 1, ..., C$ as $(\neg x_{ic} \vee \neg x_{jc})$. Also, constraints are needed for assuring that at least one color is assigned to a node, and it is given for node i as $(x_{i1} \vee x_{i2} \vee ... \vee x_{iC})$. Note that this encoding allows for more than one color being assigned to a node, so a solution of the SAT instance may yield several solutions of the original graph coloring instance. The transformation into a SAT instance yields an increase in the size of the state space from C^N to 2^{NC}.

The gsat+walk algorithm starts in a random state and uses two types of local moves (variable flips) for updates. The first is a greedy move where the variable that results in

[6] `http://web.cs.ualberta.ca/~joe/Coloring/`

the least number of broken constraint is flipped (ties are randomly broken). The second is a constrained random walk move, where a randomly chosen variable appearing in at least one broken clause is flipped.

We have used a program written by B. Selman and H. Kautz, that is available from SATLIB[7]. Different values of the parameter for the fraction of moves that are random walk moves, p, are tested to optimize the algorithm. If random move flips are included ($p \neq 0$) we have found that a single run with more flips is more beneficial than restarts with fewer flips. We have used a p-value of 0.5 and allowed for $60 \times CN$ flips.

4.5 Results

Here follows a discussion and evaluation of the testbed results for INN and the comparison algorithms.

In figures 3 – 7 we present, for each algorithm, (A) the fraction unsolved problems, f_U, as a function of edge density γ for different problem sizes, and (B) the average CPU time used as a function of problem size N, for different edge densities.

The time presented is the total time used, including reruns, until a solution is found or the maximum allowed number of reruns is done.

Parameter settings are as described above, and up to ten restarts are allowed for the ANN, INN and SAU algorithms on a problem. For the faster DSATUR algorithm up to 80 restarts are allowed, and for gsat+walk one run is used as discussed above. All experiments have been made on a 800 MHz AMD Athlon computer running Linux.

4.5.1 Discussion

As can be seen in figures 3 – 7, each algorithm seems to show a more or less distinct critical γ, above which it fails to find solutions. In all cases it is situated below the established value of $\gamma_c \approx 4.6$, and can be used as a measure of the performance. This indicates that INN and SAU are the best algorithms for difficult problems (for lower γ, where all problems are solved by either method, DSATUR wins on speed).

A closer look at the INN and SAU results reveals a tendency for SAU to perform slightly better in the lower γ range, while INN seems to have the upper hand for higher γ. To some extent, this is an effect of the chosen amount of CPU time allowed, and a different

[7] `http://www.satlib.org`

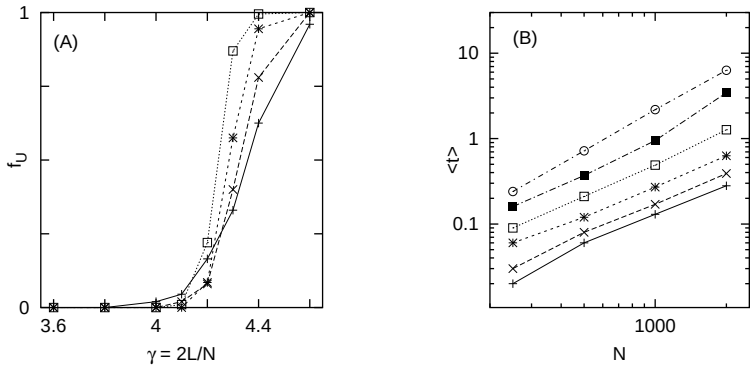

Figure 3: INN results from 200 instances for each N and γ. (**A**) Fraction unsolved problems (f_U) versus γ, for $N = 250$ (+), 500 ($\times$), 1000 ($*$), 2000 ($\Box$). The statistical error in each point is less than 0.035. (**B**) Average CPU time (in seconds) used versus N, for $\gamma = 3.6$ (+), 3.8 ($\times$), 4.0 ($*$), 4.2 ($\Box$), 4.4 ($\blacksquare$) and 4.6 ($\circ$).

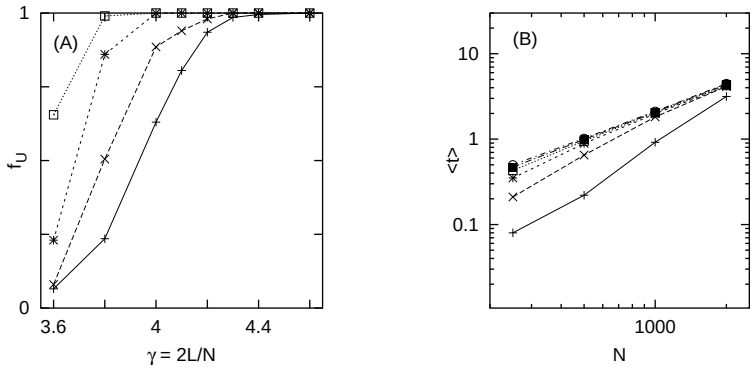

Figure 4: Conventional ANN results. (**A**) Fraction unsolved problems (f_U) versus γ. (**B**) Average CPU time (in seconds) used versus N. Notation as in figure 3.

choice might slightly change the outcome; SAU in particular, and to some extent INN, would benefit from a slower annealing rate, requiring more time.

Gsat+walk seems to be outperformed on this testset by INN and SAU; this is probably at least partly due to the disadvantageous transformation to a SAT problem. In [13] we showed that gsat+walk and INN were comparable on a number of hard 3-SAT problems, and a decrease in performance is probable also if the boolean version of INN is applied to graph coloring problems transformed into SAT. Probably a problem-specific version of gsat would perform better on this testset of graph coloring instances.

As for CPU time consumption, DSATUR seems to be very fast in solving the easier problems for small N, although the differences in time may be somewhat overestimated

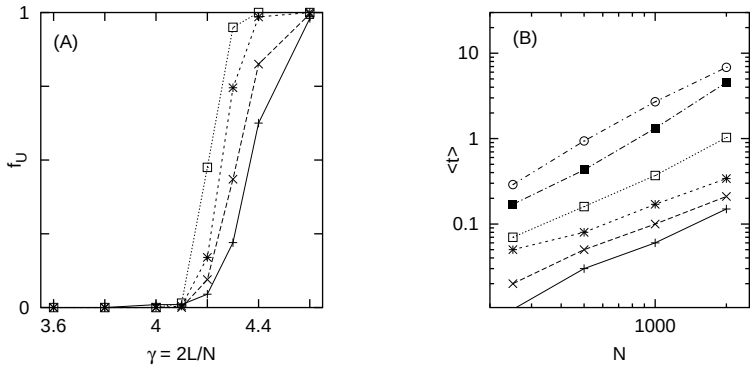

Figure 5: SAU results. (**A**) Fraction unsolved problems (f_U) versus γ. (**B**) Average CPU time (in seconds) used versus N. Notation as in figure 3.

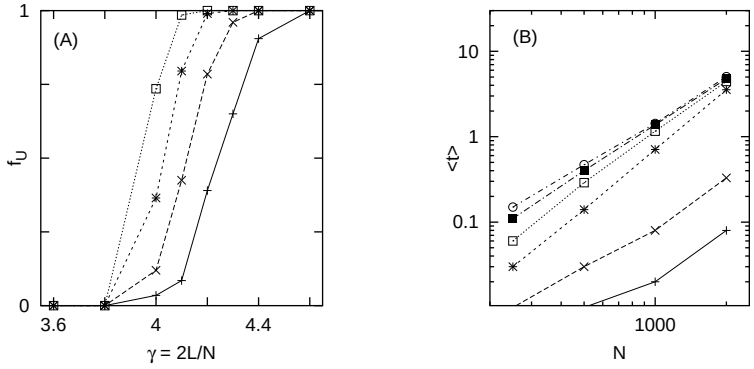

Figure 6: DSATUR results. (**A**) Fraction unsolved problems (f_U) versus γ. (**B**) Average CPU time (in seconds) used versus N. Notation as in figure 3.

due to the finite time resolution. On the other hand, the DSATUR time rises faster with increasing N, and for large N the time consumption is comparable to that of the other algorithms.

Our overall conclusion is that INN and SAU have comparable performances and are the overall winners on this testset. They are consistently superior to ANN and gsat+walk, and beat DSATUR for the large/difficult problems, where it matters the most.

4.5.2 Dense Graphs

The INN algorithm was also applied to a small set of more dense graphs available from DIMACS. The results for INN is shown in table 2. In this case we have used the stochastic

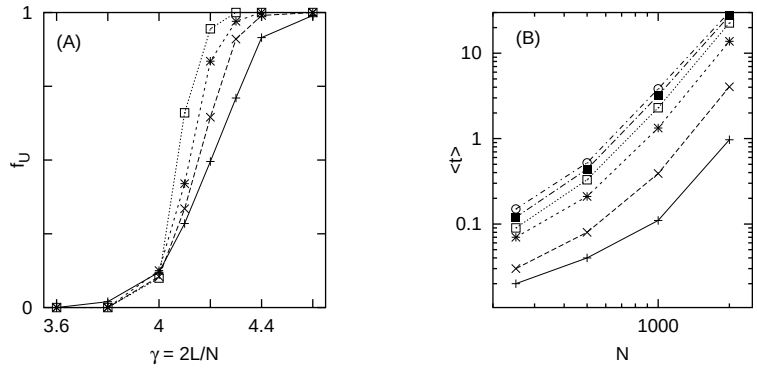

Figure 7: Gsat+walk results. (**A**) Fraction unsolved problems (f_U) versus γ. (**B**) Average CPU time (in seconds) used versus N. Notation as in figure 3.

Problem	col	E_{min}	$< E >$	E_{max}	t_{min}	$< t >$	t_{max}
g125-17/18	17	2	3.5	5	7.39	13.42	18.88
g125-17/18	18	0	0.1	1	0.26	2.99	8.98
g250-15	15	0	0	0	0.42	0.53	0.57
g250-29	29	2	3.8	5	126.25	161.36	203.48
g250-29	30	0	0.8	2	7.36	80.73	137.62

Table 2: Results for INN on some dense graphs. E counts the number of broken constraints, and t is the time measured in seconds.

version of the algorithm, and set the stop temperature to 0.1 instead of 0.3 (to allow for a longer search in the low temperature region), but otherwise the parameters are exactly the same as for the 3-coloring testbed. For the cases when a solution with optimal number of colors is not found by the algorithm, also the problem with an additional color is tried.

Both GSAT and SAU have been shown to solve these problems [11, 5], but the time used for the problems not solved by INN is quite long. For the parameter values used in this survey (with a limited maximal time) neither algorithm solved the problems that INN did not solve.

5 Summary and Conclusions

We have presented a modified ANN annealing heuristic, INN, applicable to generic CSP and generalizing a previously described method restricted to boolean CSP. It is based on an analysis of the balance of information between constraints and variables in a mean

field approximation, yielding a very specific non-polynomial cost function.

The method has been applied to a testbed of random graph 3-coloring problems, and the performance was shown to be in parity with a good dedicated heuristic, SAU, and much superior to that of a conventional ANN mean-field annealing approach based on a polynomial cost function.

The improvement compared to traditional ANN can be attributed to the strong non-linearity of the particular cost function used in INN, which boosts the ability to recognize and avoid bad solutions on an early stage, and yields an improved revision capability.

The method shares with ANN the appealing feature of not being tailored for a specific application, and can be applied to generic constraint satisfaction problems.

For constrained optimization problems, we suggest a hybrid method, where the constraints are handled by a non-linear information-based cost function, while the object function *per se* is treated with a traditional polynomial ANN cost function. Work to explore this avenue is in progress.

Acknowledgements

Thanks to D. Blancenbecler, M. Ohlsson and C. Peterson for stimulating discussions. This work was in part supported by the Swedish Foundation for Strategic Research.

References

[1] H. M. Adorf and M. D. Johnston. A discrete stochastic neural network algorithm for constraint satisfaction problems. *Proceedings of the International Joint Conference on Neural Networks*, 3:917–924, 1990.

[2] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[3] M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, 1987.

[4] J. C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In Johnson and Trick [12], pages 245–284.

[5] A. J. Davenport, E. P. K. Tsang, C. J. Wang, and K. Zhu. Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. *Proceedings of AAAI-94*, 1:325–330, 1994.

[6] D. Du, J. Gu, and P. M. Pardalos, editors. *Satisfiability Problem: Theory and Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.

[7] L. Gislén, B. Söderberg, and C. Peterson. Complex scheduling with potts neural networks. *Neural Computation*, 4(6):805–831, 1992.

[8] T. Hogg, B. A. Hubermann, and C. P. Williams. Special volume on frontiers in problem solving: Phase transitions and complexity. *Artificial Intelligence*, 81(1,2), 1996.

[9] Tad Hogg. Applications of statistical mechanics to combinatorial search problems. In D. Stauffer, editor, *Annual Reviews of Computational Physics*, volume 2, pages 357–406. World Scientific, 1995.

[10] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[11] D. S. Johnson, C. R. Aragon, and L. A. McGeoch. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partition. *Operations Research*, 39(3):378–406, 1991.

[12] D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challange, 1993, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[13] H. Jönsson and B. Söderberg. An information based neural approach to constraint satisfaction. *Neural Computation*, 13:1827–1838, 2001.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[15] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(6740):133–137, 1999.

[16] M. Ohlsson, C. Peterson, and B. Söderberg. Neural networks for optimization problems with inequality constraints - the knapsack problem. *Neural Computation*, 5(2):331–339, 1993.

[17] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[18] G. Parisi. *Statistical Field Theory*. Addison-Wesley Publishing Company, Reading, MA, 1988.

[19] C. Peterson and B. Söderberg. Neural optimization. In M. A. Arbib, editor, *The Handbook of Brain Research and Neural Networks, (2nd edition)*, pages 617–622. Bradford Books/The MIT Press, Cambridge, Massachusetts, 1998.

[20] B. Selman, H. A. Kautz, and B.Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 337–343, Menlo Park, California, 1994. AAAI Press.

[21] F. Y. Wu. The potts model. *Review of Modern Physics*, 54(1):235–268, 1982.