Finding Least Cost Proofs Using a Hierarchical PSO

Shawn T. Chivers, Gene A. Tagliarini
Dept. of Computer Science
University of North Carolina Wilmington

Ashraf M. Abdelbar
Dept. of Computer Science
American University in Cairo

Abstract— **Abduction is the process of proceeding from data describing a set of observations or events, to a set of hypotheses which best explains or accounts for the data. Cost-based abduction (CBA) is a formalism in which evidence to be explained is treated as a goal to be proven, proofs have costs based on how much needs to be assumed to complete the proof, and the set of assumptions needed to complete the least-cost proof are taken as the best explanation for the given evidence. In this paper, we explore using a hierarchical PSO to find least-cost proofs in cost-based abduction systems, comparing performance to simulated annealing using a difficult problem instance.**

I. Introduction

Cost-based abduction (CBA) [11], [12] is an important problem in reasoning under uncertainty. Finding Least-Cost Proofs (LCP's) for CBA systems is known to be $\mathcal{NP}$-hard [2], [12] and has been a subject of considerable research over the past decade. In this paper, we apply a hierarchical PSO [20], [21] to cost-based abduction. We compare performance to simulated annealing, using a 300-hypothesis, 900-rule CBA instance, that was systematically generated to be difficult.

Abduction is the process of proceeding from data describing observations or events, to a set of hypotheses, which best explains or accounts for the data [17]. A CBA system is a knowledge representation in which a given world situation is modeled as a 4-tuple $\mathcal{K} = (\mathcal{H}, \mathcal{R}, c, \mathcal{G})$, where

- $\mathcal{H}$ is a set of *hypotheses* or *propositions*,
- $\mathcal{R}$ is a set of *rules* of the form

$$(h_{i_1} \wedge h_{i_2} \wedge \ldots \wedge h_{i_n}) \longrightarrow h_{i_q} ,$$

  where $h_{i_1}, \ldots, h_{i_n}$ (called the *antecedents*) and h_{i_q} (called the *consequent*) are all members of $\mathcal{H}$,
- c is a function $c : \mathcal{H} \rightarrow \Re^+$, where $c(h)$ is called the *assumability cost* of hypothesis $h \in \mathcal{H}$ and $\Re^+$ denotes the positive reals,
- $\mathcal{G} \subseteq \mathcal{H}$ is called the *goal set* or the *evidence*.

The objective is to find the least cost proof (LCP) for the evidence, where the cost of a proof is taken to be the sum of the costs of all hypotheses that must be assumed in order to complete the proof. Any given hypothesis can be made true in two ways: it can be assumed to be true, at a cost of its assumability cost, or it can be proved. If a hypothesis occurs as the consequent of a rule R, then it can be proved, at no cost, to be true by making all the antecedents of R true, either by assumption or by proof. If a hypothesis does not appear as the consequent of any rule, then it cannot be proved, it can be made true only by being assumed. The cost of a hypothesis can be ∞, which means that it cannot be assumed, it can only be proved. One can assume, without loss of generality, that any hypothesis that appears as the consequent of any rule has an infinite assumability cost. Suppose x_q has a finite assumability cost of a, and appears as the consequent of at least one rule. One can add a hypothesis x'_q with assumability cost a, set the assumability cost of x_q to ∞, and add the rule

$$x'_q \longrightarrow x_q .$$

Therefore, we consider the hypothesis set $\mathcal{H}$ to be partitioned into two subsets: a set of assumable hypotheses $\mathcal{H}^A$, which have finite assumability costs and do not appear as consequents of any rules, and a set of provable hypotheses $\mathcal{H}^P$, which have infinite assumability costs and, hence, can be made true only by being proved.

II. Previous Work on Cost-Based Abduction

Finding an LCP for an instance of CBA was shown to be $\mathcal{NP}$-hard in 1994 [12], and even approximating an LCP within a fixed ratio bound of the optimum has recently been shown to be $\mathcal{NP}$-hard [2].

A number of approaches to this problem have been explored. Charniak and Shimony [11], [12] presented a best-first heuristic search approach, and Charniak and Husain [10] presented an admissible heuristic for the problem.

Santos [32], [33] presented a method for transforming a CBA instance into a set of linear constraints, which could then be solved by 0-1 integer linear programming (ILP). Santos' operations research based approach was followed by several others: Ishizuka and Matsuo [19] presented a method called *slide down and shift up*, which uses a combination of linear programming and nonlinear programming to find near-optimal solutions in polynomial-time; Ohsawa and Ishizuka [30] presented a method called *bubble propagation*, which also finds near-optimal solutions in polynomial-time; Matsuo and Ishizuka [28] investigated linear and nonlinear programming approaches to CBA and to more general logical reasoning problems such as satisfiability. Santos and Santos [34] presented sufficient conditions for a CBA instance to be polynomially-solvable based on the idea of totally unimodular matrices; their work has been extended by Ohsawa and Yachida [31].

Abdelbar [1] showed that methods for cost-based abduction can be used for belief revision on belief networks. Kato *et al.* [22] investigated a method for finding LCP's based on binary decision diagrams. Den [14] presented a chart-based method for cost-based abduction. Kato *et al.* [23] investigated a search control mechanism for the A^* algorithm for cost-based abduction, and Kato *et al.* [24] investigated the parallelization of cost-based abduction with parallel best-first search. Recently, neural network [6], [8], ant colony [9], iterated local search [7], and population-oriented simulated annealing approaches [4], [5] to cost-based abduction have also been explored.

III. PSO AND HIERARCHICAL PSO

Particle swarm optimization (PSO) [15], [25], [26], [27] is a computational paradigm based on the phenomenon of collective intelligence exhibited by swarms of insects, schools of fish, and herds of buffalo, that has been applied to a variety of domains [16]. PSO can be used for both discrete and continuous optimization problems; we restrict our attention here to PSO as a discrete optimization technique.

Computation in the PSO paradigm is based on a collection, called a *swarm*, of fairly-primitive processing elements, called *particles*. Connectivities are defined over the swarm, whereby each particle has an adjacency relationship with a subset of the other particles in the swarm. The neighborhood of each particle is the set of particles with which it is adjacent; in standard PSO. Two well-studied neighborhood structures are *gbest*, in which the entire swarm is considered a single neighborhood, and *lbest*, in which the particles are arranged in a ring, and each particle's neighborhood consists of itself, its immediate ring-neighbor to the right, and its immediate ring-neighbor to the left.

Suppose we would like to use a swarm of M particles to solve a discrete combinatorial optimization problem whose candidate solutions can be represented as vectors of bits; let $\mathcal{I}$ be a given instance of such a problem. Let N denote the number of elements in the solution vector for $\mathcal{I}$. Each particle i would contain two N-dimensional vectors: a boolean state vector x_i, which represents a candidate solution to $\mathcal{I}$ and is called particle i's state, and a real vector v_i, called the *velocity* of the particle. In the biological insect-swarm analogy, the velocity vector represents how fast, and in which direction, the particle is flying for each dimension of the problem being solved.

Let $\mathcal{N}(i)$ denote the neighbors of particle i, and let p_i denote the best solution ever found by particle i. In each time iteration, each particle i adjusts its velocity based on a weighted stochastic average of

- its previous velocity,
- the difference between its current state and the best state so far of its neighbors, and
- the difference between its current state and its own best state so far.

Specifically, the new velocity vector is determined by

$$v_i = \alpha v_i^{old} + \phi_1 rand()(p_i - x_i) + \phi_2 rand()(p_g - x_i) \ , \quad (1)$$

where

- α, called *inertia*, is a parameter within the range $[0, 1]$ and is often decreased over time [35],
- ϕ_1 and ϕ_2 are two non-negative constants, often chosen so that $\phi_1 + \phi_2 = 4$ [27], which control the degree to which the particle "follows the herd" thus stressing exploitation (higher values of ϕ_2), or "goes its own way" thus stressing exploration (higher values of ϕ_1),
- $rand()$ is a uniformly random number generator function that returns values within the range $(0, 1)$,
- g is the particle in i's neighborhood with the current neighborhood-best candidate solution.

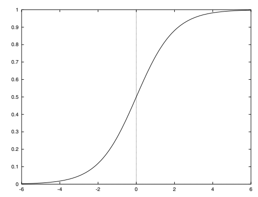

Fig. 1. The sigmoid function

For each dimension $j = 1, \ldots, N$, we then apply

$$v_{ij} = \begin{cases} V_{max} & \text{if } v_{ij} > V_{max} \\ v_{ij} & \text{if } -V_{max} \leq v_{ij} \leq V_{max} \\ -V_{max} & \text{if } v_{ij} < -V_{max} \end{cases} \quad (2)$$

where V_{max} is a constant that limits the growth of velocity in either direction; then, we apply

$$Pr(x_{ij} = 1) = \sigma(v_{ij}) \ , \quad (3)$$

where σ is the sigmoidal function $\sigma(a) = \frac{1}{1+e^{-a}}$, shown in Figure 1. Equation (3) can be implemented by generating a random number ρ within the range $(0, 1)$ and setting x_{ij} to 1 if $\rho < \sigma(v_{ij})$ and to 0 otherwise.

We then determine the highest-quality solution within each neighborhood, according to a domain-dependent fitness measure, and adjust each particle's best neighbor pointer accordingly.

In all, the parameters that need to be adjusted in the standard PSO system we have described are:

- the inertia α and its decrease schedule,
- the coefficients ϕ_1 and ϕ_2,
- the limiting value V_{max}, and
- the neighborhood topology.

A number of variations on the standard PSO described above have been explored. These include the fully-informed PSO introduced by Mendes, Kennedy, and Neves [29], dynamical neighborhood structures investigated by Hu and Eberhart [18], the use of mutation in PSO studied by Stacey, Jancic, and Grundy [36], and the incorporation in PSO of a heuristic "goodness" function similar to the η function used in ant systems [3]. Recent work on the convergence properties of PSO includes Clerc and Kennedy [13], Trelea [37], and van den Bergh and Engelbrecht [38].

Introduced in 2003, hierarchical PSO (HPSO) [20], [21] is a variation on the original technique; in HPSO, particles are arranged in a tree-based topology (as in, e.g., Figure 2), in which the better-performing particles float towards the top of the tree. In each iteration, starting from the root of the tree, and moving downwards in breadth-first fashion, each particle

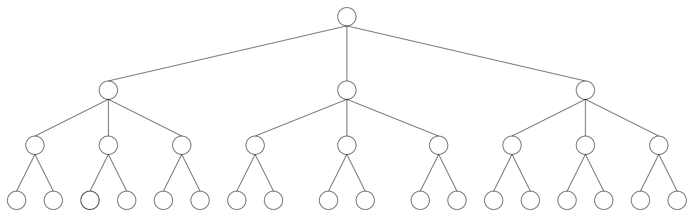

Fig. 2. An example of a hierarchical PSO topology, with height $h = 4$, maximum degree $d = 3$, and number of nodes $m = 31$.

i is compared to its immediate children. If at least one of the children is better than i, then the best of the children is swapped with i. The breadth-first movement means that, in a single iteration, a particle can move upwards only one level in the tree but can move downwards many levels. In HPSO, Equation (1) continues to apply, except that g now represents particle i's immediate parent in the tree.

The topology of the tree used in HPSO is defined by three parameters: the height h, the node degree d, and the total number of nodes m. All nodes in the tree have the same degree except for the level immediately above the leaf nodes in which nodes might have a degree less than d; for that level, the leaves are distributed so that the maximum difference in degree in the nodes at that level is at most 1.

IV. Applying PSO to CBA

In modeling CBA problems with PSO, we let the number of dimensions equal the number of assumable hypotheses $|\mathcal{H}^A|$. A value of 1, or 0, for a given dimension indicates whether a hypothesis is assumed or not, respectively. Thus, the x_i state vector for particle i represents a candidate assignment to the assumable hypotheses. Of course, such an assignment will not necessarily be sufficient to prove the goal, i.e. will not necessarily represent a feasible solution.

Many approaches are possible for handling these unfeasible solutions. One approach, used in [9] in applying an ant colony approach to CBA, is to simply leave the unfeasible solution as it is, and to make its fitness equal to the cost of the hypotheses assumed plus a fixed, large penalty. Another approach is to use a heuristic repair technique to modify the unfeasible solution so that it proves the goal. In [7], an elaborate repair technique was used to produce a heuristic estimate of the cost of the hypotheses that would need to be added to the assumable hypotheses in order to prove the goal; this estimate was then used as the fitness of the solution, but the search process proceeded based on the unfeasible solution, i.e. these additional hypotheses were used to estimate the fitness of the solution but were not actually added to the hypotheses assumed.

In this paper, we use a repair technique based on a type of stochastic local search. At the end of each iteration, the x_i vector of each particle is examined. If the hypotheses assumed are sufficient to prove the goal, then the fitness of the solution is made equal to the assumability cost of the hypotheses corresponding to the 1-bits of x_i and no further processing is needed. Otherwise, we randomly choose a 0-bit in the x_i vector and assign it to 1. If the goal still cannot be proven, then we randomly choose another 0-bit and assign it to 1, until the goal is provable. This process can of course result in many unnecessary hypotheses being assumed. We, therefore, follow up this process with a simple 1-OPT optimization process. We examine each of the 1-bits of the x_i vector: one by one and in a random order, each 1-bit is assigned to 0 and if the goal can still be proven, then it is retained as 0, otherwise it is set back to 1.

To create the initial population, the x_i vectors of all particles are randomly assigned, then the repair process described above is applied to each x_i vector.

TABLE I

Basic characteristics of the CBA instance `raa180` (available at **www.cbalib.org**), used in our experimental results.

Instance	`raa180`
No. of hypotheses	300
No. of rules	900
No. of assumable hypotheses	180
Rule depth	max: 38, avg: 25.0 median: 27
No. of rules in which a single hypothesis appears as a consequent	max: 15, avg: 7.5 median: 7
No. of rules in which a single hypothesis appears as an antecedent	max: 72, avg: 14.6 median: 11
Optimal solution cost	10,821
ILP CPU time (sec)	88,835
ILP tree depth	41
ILP nodes	178,313

V. Experimental Results

In [8], we explored the issue of generating difficult instances of CBA. We used a CBA random instance generator which takes as parameters the total number of hypotheses, the number of rules, and a lower bound on the number of assumable hypotheses. As described in [8], initial experimentation suggested that a ratio of 1:3 for the total number of hypotheses relative to the number of rules yielded difficult instances.

In one experiment, we fixed the total number of hypotheses at 300, and the number of rules at 900, and allowed the number of assumable hypotheses to vary from 40 to 200, in steps of 10, generating 25 random instances at each step. Thus, the total number of CBA instances generated was 425 (these are available at **www.cbalib.org** as `Collection A`). Out of these 425 instances, a method was needed to choose the most likely to be difficult without exactly solving each instance. Each instance was converted to an integer linear program (ILP) using Santos' method [33]. The linear program (LP) relaxation of the ILP corresponding to each instance was then solved using the popular public-domain engine `lp-solve`. The number of non-integral variables in the solution to the linear program, denoted f, was determined. The worst-case run-time of the full ILP is bounded from above by 2^f, although, of course, because of the branch-and-bound pruning process, the actual run-time may not necessarily be a function

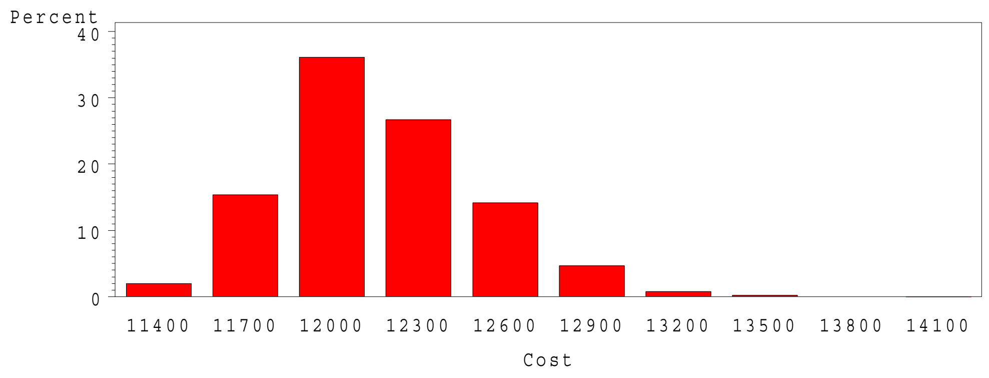

Fig. 3. Distribution of HPSO solutions for `raa180`

TABLE II

STATISTICS OF SOLUTION DISTRIBUTION FOR HPSO ON `raa180`

Statistic	Value
No. of trials	3,584
Mean (μ) Solution Cost	12,155.00
Std Dev (σ) Solution Cost	350.35
Ratio σ/μ for Solution Cost	0.029
Median Solution Cost	12,119
Mean Time (secs)	125.88
Std Dev Time (secs)	54.55

TABLE III

COMPARING HPSO ON `raa180` WITH HEURISTIC REPAIR VERSUS WITH A CONSTANT PENALTY

	Penalty	Repair	Ratio
Mean (μ) Solution Cost	14,991.27	12,155.00	123.3%
Std Dev (σ) Solution Cost	843.16	350.35	240.7%
Ratio σ/μ for Solution Cost	0.06	0.03	195.1%
Mean (μ) Time (secs)	6.52	125.88	5.2%
Std Dev (σ) Time (secs)	0.93	54.55	1.7%
Ratio σ/μ for Time (secs)	0.14	0.43	32.8%

of f. However, lacking another measure, we used the ratio of f to the total number of variables as a rough predictor of the difficulty of the problem instance. Based on this ratio, we chose the problem instance, `raa180`, which had the highest value for this ratio. The optimal solution for `raa180` was obtained through Santos' ILP method, using `lp-solve` as the ILP engine. Table I shows some characteristics of this instance as well as the run-time taken by `lp-solve`, and the depth and the number of nodes of the ILP branch-and-bound tree processed by `lp-solve`. The data in Table I is obtained from [8, Table I]

Using `raa180` as the benchmark CBA instance, we applied a hierarchical PSO with height $h = 3$, degree $d = 5$, and number of particles $m = 31$. Based on [37] and following [21], we used $\phi_1 = \phi_2 = 1.494$, $V_{max} = 6$, and an inertia factor α that starts at 0.729 and decreases to 0.4. We executed

TABLE IV

STATISTICS OF SOLUTION DISTRIBUTION FOR SA ON `raa180`

Statistic	Value
No. of trials	10,247
Mean (μ) Solution Cost	11,184.58
Std Dev (σ) Solution Cost	289.25
Ratio σ/μ for Solution Cost	0.026
Mean SA Iterations	75,971.15
Std Dev SA Iterations	110,860.7
Mean Time (secs)	112.72
Std Dev Time (secs)	164.28

more than 3,500 trials, and allowed the HPSO to run for 500 iterations in each trial (this was sufficient for convergence). Figure 3 shows the distribution of solutions that was obtained. We can see that the best solution observed over all trials had a cost only 5% higher than the optimal, but was observed in only about 2% of the trials. The solution most frequently observed, in about 35% of the trials, had a cost approximately 11% higher than the optimal. As Table II shows, the mean solution cost was 12% higher than the optimal, and the ratio of the standard deviation to the mean solution cost is very small.

To isolate the effect of the heuristic repair function, we also executed 100 trials without heuristic repair and 1-OPT search; instead, the fitness of unfeasible solutions was simply made equal to the cost of the hypotheses assumed plus a fixed, large penalty. Table III shows these results in comparison to the results using heuristic repair. We observe that the mean solution cost, as might be expected, is significantly worse, and the ratio σ/μ is considerably higher without repair. On the other hand, the heuristic repair function results in nearly a twenty-fold increase in CPU time. We also observe that, with the penalty method, the variation in the CPU time is much smaller.

We compare our results to simulated annealing (SA) using data from [7]. Table IV shows the results of running more than 10,000 trials of SA on `raa180`. These were based on the heuristic repair function described in [7], with a temperature decrease factor of 0.999, and an initial temperature T_0 that was set so as to make

$$e^{-\frac{\hat{\Delta}}{T_0}} = 0.5 \ , \tag{4}$$

where $\hat{\Delta}$ is the average difference in cost between neighboring states. In the case of `raa180`, this resulted in $T_0 = 720$. Each SA trial was allowed to run until 100 iterations elapsed without change in solution.

Although the SA algorithm performed significantly better than the HPSO algorithm in our experiment, there is reason to believe that the basic HPSO algorithm can reach similar performance levels with further improvements.

VI. Conclusion & Future Directions

In this paper, we applied a hierarchical PSO to a difficult instance of CBA, and found that the HPSO had performance comparable to simulated annealing. We expect that performance of the HPSO can be improved by using the adaptive variant of HPSO [21], in which the height of the HPSO's tree topology is gradually increased over the course of the computation, and the degree d is gradually decreased.

Another possibility is a hybrid in which HPSO and SA are run as two concurrent threads. Periodically, a particle i is selected from the bottom level of the HPSO tree, and its current state vector x_i is replaced with the current solution vector of the SA.

References

[1] A.M. Abdelbar, "An algorithm for finding MAP explanations through cost-based abduction," *Artificial Intelligence*, Vol. 104, pp. 331-338, 1998.

[2] Ashraf M. Abdelbar, "Approximating cost-based abduction is NP-hard," *Artificial Intelligence*, Vol. 159, No.1-2, pp. 231-239, November 2004.

[3] A.M. Abdelbar, and S. Abdelshahid, "Swarm optimization with instinct-driven particles," *Proceedings IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 777-782, 2003.

[4] A.M. Abdelbar, and H. Amer, "Applying guided evolutionary simulated annealing to cost-based abduction," *Proceedings IEEE International Joint Conference on Neural Networks*, Vol. 3, pp. 2428-2431, 2003,.

[5] A.M. Abdelbar, Heba A. Amer, "Finding least-cost proofs with population-oriented simulated annealing," *Proceedings Conference on Artificial Neural Networks in Industrial Engineering (ANNIE-06)*, pp. 79-84, November 2006.

[6] A.M. Abdelbar, E.A.M. Andrews, and D.C. Wunsch, "Abductive reasoning with recurrent networks," *Neural Networks*, Vol. 16, No. 5-6, pp. 665-673, 2003.

[7] A.M. Abdelbar, S.H. Gheita, and H. Amer, "Exploring the fitness landscape and the run-time behavior of an iterated local search algorithm for cost-based abduction," *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 18, No. 3, pp. 365-382, September 2006.

[8] A.M. Abdelbar, M.A. El-Hemaly, E.A.M. Andrews, and D.C. Wunsch, "Recurrent neural networks with backtrack-points and negative reinforcement applied to cost-based abduction," *Neural Networks*, Vol. 18, pp. 755-764, August 2005.

[9] A.M. Abdelbar, and M. Mokhtar, "A k-elitist MAX-MIN ant system approach to cost-based abduction," *Proceedings IEEE Congress on Evolutionary Computation*, Vol. 4, pp. 2635-2641, 2003.

[10] E. Charniak, and S. Husain, "A new admissible heuristic for minimal-cost proofs," *Proceedings AAAI National Conference on Artificial Intelligence*, pp. 446-451, 1991.

[11] E. Charniak, and S.E. Shimony, "Probabilistic semantics for cost-based abduction," *Proceedings AAAI National Conference on Artificial Intelligence*, pp. 106-110, 1990.

[12] E. Charniak, and S.E. Shimony "Cost-based abduction and MAP explanation," *Artificial Intelligence*, Vol. 66, pp. 345-374, 1994.

[13] M. Clerc and J. Kennedy, "The particle swarm—explosion, stability, and convergence in multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 1, pp. 58-73, 2002.

[14] Y. Den, "Generalized chart algorithm: an efficient procedure for cost-based abduction," *Proceedings Meeting of the Association for Computational Linguistics*, pp. 218-224, 1994.

[15] R.C. Eberhart, and J. Kennedy, "A new optimizer using particle swarm theory," In *Proceedings International Symposium on Micro Machine and Human Science*, pp. 39-43, 1995.

[16] R.C. Eberhart, and Y. Shi, "Particle swarm optimization: developments, applications and resources," *Proceedings IEEE International Conference on Evolutionary Computation*, pp. 81-86, 2001.

[17] J.R. Hobbs, M.E. Stickel, D.E. Appelt, and P. Martin, "Interpretation as abduction," *Artificial Intelligence*, Vol. 63, pp. 69-142, 1993.

[18] X. Hu, and R.C. Eberhart, "Multiobjective optimization using dynamic neighborhood particle swarm optimization," *Proceedings Congress on Evolutionary Computation*, pp. 1677-1681, 2002.

[19] M. Ishizuka, and Y. Matsuo, "SL method for computing a near-optimal solution using linear and non-linear programming in cost-based hypothetical reasoning," *Proceedings Pacific Rim International Conference on Artificial Intelligence*, pp. 611-625, 1998.

[20] S. Janson, and M. Middendorf, "A hierarchical particle swarm optimizer," *Proceedings IEEE Congress on Evolutionary Computation*, 2003.

[21] S. Janson, and M. Middendorf, "A hierarchical particle swarm optimizer and its adaptive variant," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, Vol. 35, No. 6, December 2005.

[22] S. Kato, S. Oono, H. Seki, and H. Itoh, "Cost-based abduction using binary decision diagrams," *Proceedings Industrial and Engineering Applications of Artificial Intelligence*, pp. 215-225, 1999.

[23] S. Kato, H. Seki, and H. Itoh, "Cost-based horn abduction and its optimal search," *Proceedings Third International Conference on Automation, Robotics and Computer Vision*, pp. 831-835, 1994.

[24] S. Kato, H. Seki, and H. Itoh, "Parallel cost-based abductive reasoning for distributed memory systems," *Proceedings Pacific Rim International Conference on Artificial Intelligence*, pp. 300-311, 1996.

[25] J. Kennedy, and R.C. Eberhart, "Particle swarm optimization," In *Proceedings IEEE International Conference on Neural Networks*, Vol. IV, pp. 1942–1948, 1995.

[26] J. Kennedy, and R.C. Eberhart, "A discrete binary version of the PSO algorithm," *Proceedings IEEE Conference on Systems, Man and Cybernetics*, pp. 4104-4109, 1997.

[27] J. Kennedy, and R. C. Eberhart, *Swarm Intelligence*, Morgan Kaufmann, San Francisco, 2001.

[28] Y. Matsuo, and M. Ishizuka, "Two transformations of clauses into constraints and their properties for cost-based hypothetical reasoning," *Proceedings Pacific Rim Conference on Artificial Intelligence*, pp. 118-127, 2002.

[29] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: simpler, maybe better," *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 4, pp. 204-210, 2004.

[30] Y. Ohsawa, and M. Ishizuka, "Networked bubble propagation: A polynomial-time hypothetical reasoning method for computing near-optimal solutions," *Artificial Intelligence*, Vol. 91, pp. 131-154, 1997.

[31] Y. Ohsawa, and M. Yachida, "An extended polynomial solvability of cost-based abduction," *Proceedings International Joint Conference on Artificial Intelligence (poster session abstracts)*, p. 79, 1997.

[32] E. Santos Jr., "On the generation of alternative explanations with implications for belief revision," *Proceedings Seventh Conference on Uncertainty in AI*, pp. 339-347, 1991.

[33] E. Santos Jr., "A linear constraint satisfaction approach to cost-based abduction," *Artificial Intelligence*, Vol. 65, pp. 1-27, 1994.

[34] E. Santos, Jr., and E.S. Santos, "Polynomial solvability of cost-based abduction," *Artificial Intelligence*, Vol. 86, pp. 157-170, 1996.

[35] Y. Shi, and R.C. Eberhart, "A modified particle swarm optimizer," *Proceedings IEEE International Conference on Evolutionary Computation*, pp. 69-73, 1998.

[36] A. Stacey, M. Jancic, and I. Grundy, "Particle swarm optimization with mutation," *Proceedings IEEE Congress on Evolutionary Computation*, 2003.

[37] I.C. Trelea, “The particle swarm optimization algorithm: convergence analysis and parameter selection,” *Information Processing Letters*, Vol. 85, pp. 317-325, 2003.

[38] F. van den Bergh, and A.P. Engelbrecht, “A new locally convergent particle swarm optimizer,” *Proceedings IEEE International Conference on Systems, Man and Cybernetics*, pp. 94-99, 2002.