Open access • Book Chapter • DOI:10.1007/978-3-540-30202-5_17

Incremental Knowledge Acquisition for Improving Probabilistic Search Algorithms

J. P. Bekmann, J. P. Bekmann, Achim Hoffmann

Institutions: University of New South Wales, NICTA

Published on: 05 Oct 2004 - Knowledge Acquisition, Modeling and Management

Topics: Probabilistic analysis of algorithms, Estimation of distribution algorithm, Incremental heuristic search, Search algorithm and Probabilistic logic

Incremental Knowledge Acquisition for Improving Probabilistic Search Algorithms

J.P. Bekmann[1,2] and Achim Hoffmann[1]

[1] School of Computer Science and Engineering,
University of New South Wales, NSW 2052, Australia.
[2] National ICT Australia (NICTA), A.T.P, NSW 1430, Australia.

Abstract. A new incremental knowledge acquisition approach for the effective development of efficient problem solvers for combinatorial problems based on probabilistic search algorithms is proposed. The approach addresses the known problem of adapting probabilistic search algorithms, such as genetic algorithms or simulated annealing, by the introduction of domain knowledge. This is done by incrementally building a knowledge base that controls parts of the probabilistic algorithm, e.g. the fitness function and the mutation operators in a genetic algorithm.
The probabilistic search algorithm is monitored by a human who makes recommendations on search strategy based on individual solution candidates. It is assumed that the human has a reasonable intuition of the search problem. The human adds rules to a knowledge base describing how candidate solutions can be improved, or characteristics of candidate solutions which he/she feels are likely or unlikely to lead to good solutions. Our framework is inspired by the idea of (Nested) Ripple Down Rules where humans provide exception rules to rules already existing in the knowledge base using concrete examples of inappropriate performance of the existing knowledge base.
We present experiments on industrially relevant domains of channel routing as well as switchbox routing in VLSI design. We show very encouraging inital experimental results demonstrating that our approach can solve problems comparably well to other approaches. These other approaches use algorithms developed over decades, while we were able to develop an effective search procedure in a very short time. A brief discussion outlines our KA experience with these experiments.

1 Introduction

General purpose search algorithms attempting hard problems rely on the introduction of domain knowledge in order to make the search feasible. For a long time Genetic Algorithms (GA) have been considered to be general purpose search techniques which can be applied to all sorts of search problems. In practice, however, it usually proves to be rather difficult to adapt a general purpose GA design to a particular problem type at hand [1, 2]. To tailor the GA to a problem really well may easily take months of development.

Anecdotal accounts suggest that tailoring a general purpose probabilistic search technique, such as GA or simulated annealing, to suit a given problem can take months or even years while the implementation of the basic algorithm can be done in a matter of days.

Hence, it is critical to address the problem of tailoring general algorithms to suit a given problem. In genetic algorithms, there are a number of issues which need to be adjusted for a problem. They include general parameters, such as number of generations, population size, the problem encoding and the way offspring is generated (i.e. what kind of mutation, cross-over etc.).

In particular, for the way of how offspring are generated it appears that a knowledge acquisition approach can be used to address the problem as humans seem generally to have some idea of what kind of offspring might improve the chances of finding a (good) solution.

We pursued the idea that a knowledge acquisition approach could be used to develop a knowledge base that controls the generation of promising offspring. While the GA without a tailored knowledge base can be expected to find some sort of solution (often by far suboptimal), after sufficient computing time is given (often excessive), an incrementally developed knowledge base for improving the offspring generation lets us expect a gradual improvement in performance of the GA.

We chose an incremental knowledge acquisition process, inspired by Ripple Down Rules [5], as it seems usually possible for a human, who has some idea of how to find solutions for the problem at hand, to judge at least for extreme cases, which offspring should be ignored and what kind of offspring are promising compared to their parents.

The incremental knowledge acquisition approach we present in this paper allows the user to inspect individuals as they were generated by the existing GA process. If a generated individual is considered to be useless the probability of it being generated can be reduced by providing suitable characteristics of the kind of offspring that should be generated less often or not at all. Similarly, the probability of promising offspring to be generated can be increased in the same way. Furthermore, if the human wants to propose a particular way of constructing an offspring from a parent (or parents), the human can define a suitable operator that modifies the parent(s) accordingly. The applicability of such a newly introduced operator would again be controlled by a set of rules, which are organised in a Ripple Down Rules (RDR) structure.

Our incremental knowledge acquisition framework ensures that previously provided knowledge about the quality of individuals is largely maintained while additional knowledge is integrated into the knowledge base by only being applicable to those cases where the existing knowledge base did not judge in accordance with the human. In other words, the adverse interaction of multiple rules in a knowledge base is effectively avoided. Our work differs is some important aspects from traditional RDR in that conditions and rules in the knowledge base (KB) can be edited, and solutions can be found with an incomplete KB (specification that does not cover all possible cases).

Our framework HeurEAKA (Heuristic Evolutionary Algorithms using Knowledge Acquisition) allows the GA to run in conjunction with the current - initially empty - knowledge base on problem instances. The evolutionary process can be monitored by the human and individuals can be evaluated. If a particular individual is generated that appears undesirable or suboptimal, the human could enter a new rule that prevents such behaviour in future or provide an improved alternative action. The user might also add a rule which imposes a fitness penalty on such individuals. More generally, the user formulates rules based on charac-

teristics of selected individuals, and these are applied in the general case by the GA.

We expect the application of KA to other GA problems to be promising whenever an expert has some kind of intuition of what differentiates good candidates from bad candidates, as well as being able to make at least a guess at how individual candidates may be improved. Given these considerations, we expect that our approach is applicable to most practical GA applications.

This paper is organised as follows: In the next section we present our knowledge acquisition framework HeurEAKA including a brief review of genetic algorithms. Section 3 presents a case study where our framework was applied to the problem of switchbox routing, an industrially relevant problem from the realm of VLSI design. It also briefly discusses earlier experiments with the problem of channel routing in VLSI design. The following sections, section 4 and section 5 discuss our results and the lessons learned so far. This is followed by the conclusions in section 6.

2 Our Incremental Knowledge Acquisition Framework HeurEAKA

The HeurEAKA framework naturally falls into a genetic algorithm and a knowledge acquisition component. The GA is essentially a general purpose genetic algorithm. The KA part comprises a knowledge base manager which controls modification of the KB as well as the evaluation of cases supplied by the GA. The KA module contains a *primitives interface* which allows customization for problem domain specific functionality. The framework is implemented with a graphical user interface, but also supports batch-style processing.

2.1 Genetic Algorithms

Evolutionary algorithms are loosely based on natural selection, applying these principles to search and optimisation. These include evolution strategy, evolutionary programming, genetic programming and genetic algorithms, which we concentrate our work on.

Basic GAs are relatively easy to implement. A solution candidate of the problem to be solved is encoded into a genome. A collection of genomes makes up a population of potential solutions. The GA performs a search through the solution space by modifying the population, guided by an evolutionary heuristic. When a suitable solution has been identified, the search terminates.

A genetic algorithm usually starts with a randomly initialized population of individuals, and searches through the solution space guided by a *fitness* value assigned to individuals in the population. Based on probabilistic operators for selection, mutation and crossover, the GA directs the search to promising areas. GAs have been applied to a wide variety of domains, and were found to be quite effective at solving otherwise intractable problems [3] .

GAs do suffer from a variety of problems, most notably the "black art" of tuning GA parameters such as population size, selection strategies, operator weightings, as well as being very sensitive to problem encoding and operator formulation [1, 4]. We aim to address some of these issues with an explicit formulation of domain knowledge using well suited KA techniques.

Genome encoding and manipulation is treated by the GA as opaque. All manipulations take place indirectly via the *primitives interface* (see Section 3.3).

In order to generate new individuals, the GA has to select parents from the current population and then generate offspring either by mutation and/or by crossover. Further, some individuals of the current generation should be selected for removal and replaced by newly generated individuals.

Offspring of selected parents are either created via a crossover copy operation or as a mutated copy. A parameter determines which operator will be applied. The crossover operator mimics natural evolutionary genetics and allows for recombination and distribution of successful solution sub-components in the population.

In order to select individuals either as a parent for a new individual or as a candidate to be removed from the population, the knowledge base is invoked to determine the fitness of an individual as explained below. In order to generate suitable offspring, another knowledge base is invoked which probabilistically selects mutation operators.

Fig. 1 illustrates the interaction between the GA and the KA module.

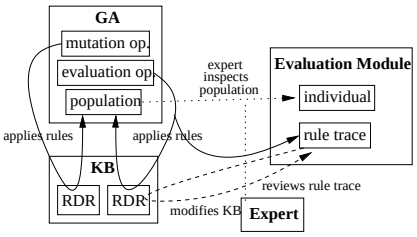

Fig. 1. The GA consults the KB when evaluating and mutating a genome. Picking an individual, the expert can iteratively review the corresponding rule trace and modify the KB. For compactness, the diagram only shows the refinement of the evaluation operators - in practice, the mutation operators are refined in the same way.

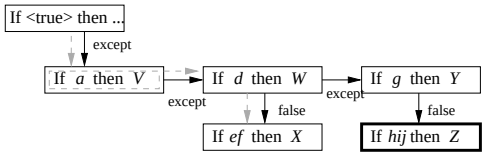

Fig. 2. A simple RDR structure. The dashed line indicates the path taken & rule executed (V) for $a\overline{de}$, the bold box indicates how action Z would be added for $ad\overline{g}hij$.

2.2 Knowledge Acquisition

Our knowledge acquisition approach for building the knowledge base for fitness determination and the knowledge base for selecting operators for offspring generation is based on the ideas of ripple down rules (RDR) [5]. RDR builds a rule base incrementally based on specific problem instances for which the user explains their choices. An extension of RDR allows hierarchical structuring of RDRs - "nested RDR" (NRDR) [6]. NRDR allows re-use of definitions in a KB, and the abstraction of concepts which make it easier to describe complex problems on the knowledge level and also allow for more compact knowledge bases for complex domains.

Ripple Down Rules Knowledge Base: We use single classification RDRs (SCRDRs) for both types of knowledge bases. A single classification RDR is a binary tree where the root node is also called the default node. To each node in the tree a rule is associated, with a condition part and a conclusion which is usually a class - in our case it is an operator application though. A node can have

up to two children, one is attached to an *except* link and the other one is attached to the so-called if-not link. The condition of the default rule in the default node is always true and the conclusion is the default conclusion. When evaluating a tree on a case (the object to be classified), a *current conclusion* variable is maintained and initialised with the default conclusion. If a node's rule condition is satisfied, then its conclusion overwrites the *current conclusion* and the except-link, if it exists, is followed and the corresponding child node is evaluated. Otherwise, the if-not link is followed, if it exists, and the corresponding child node is evaluated. Once a node is reached such that there is no link to follow the *current conclusion* is returned as a result. Figure 2 shows a simple RDR tree structure. In bold is a rule that a user might have added for the case where conditions $ad\overline{g}hij$ hold, causing action Z to be executed, instead of W.

In typical RDR implementations, any KB modification would be by adding exception rules, using conditions which only apply to the current case for which the current knowledge base is inappropriate. By doing this, it is ensured that proper performance of the KB on previous cases is maintained.

Nesting RDRs allows the user to define multiple RDRs in a knowledge base, where one RDR rule may use another, nested RDR tree in its condition, and in HeurEAKA as an action. I.e. the nested RDR tree is evaluated in order to determine whether the condition is satisfied. A strict hierarchy of rules is required to avoid circular definitions etc.

For the purpose of controlling the Genetic Algorithm, in our approach all conclusions are actually actions that can be applied to the case, which is an individual genome. The rules are formulated using the Rule Specification Language as detailed below.

Fitness Knowledge Base The user specifies a list of RDRs which are to be executed when the fitness of a genome is determined. The evaluation task can thus be broken into components as needed, each corresponding to a RDR. The evaluator executes each RDR in sequence.

Mutation Knowledge Base For determining a specific mutation operator a list of RDRs provided by the user is consulted. Each RDR determines which specific operator would be applied for modifying an individual. Unlike for the evaluation, for mutation only one of the RDRs for execution will be picked probabilistically using weights supplied by the user.

Rule Specification Language (RSL): Conditions and actions for a rule are specified in a simple language based loosely on "C" syntax. It allows logical expressions in the condition of a rule, and a list of statements in the action section.

The RSL supports variables and loop structures. Domain specific variable types can be defined, as well as built-in primitive commands relevant to the problem domain. Section 3 gives examples of rule specification.

The Knowledge Acquisition Process in HeurEAKA: The knowledge acquisition process goes through a number of iterations as follows: on each individual the fitness KB is applied and a set of rules executed. The user can select some of the individuals that appear to be of interest and then review the rules that were involved in the creating of those individuals.

The genetic algorithm can be started, stopped and reset via a graphical user interface. A snapshot of the GA population is presented, from which the user can pick an individual for closer inspection. Figure 3 shows an example of the GA window, in this case applied in the problem domain of switchbox routing.

As shown in Fig. 4, an individual solution can be inspected. It was found necessary to have a good visualization and debugging interface to be able to productively create and test rules.

A user can step back, forward and review the application of RDR rules to the genome, and make modifications to the respective KB by adding exception rules. Figure 10 shows an example of how these would be entered.

Since the evaluation of a set of RDRs can cause a number of actions to be executed, it is necessary to allow the user to step through an execution history. Given non-deterministic elements of operator selection, the interactive debugger has to maintain complete state descriptions, i.e. genome data, variable instantiation and values used in random elements, to make it feasible for the user to recreate conditions for repeated testing.

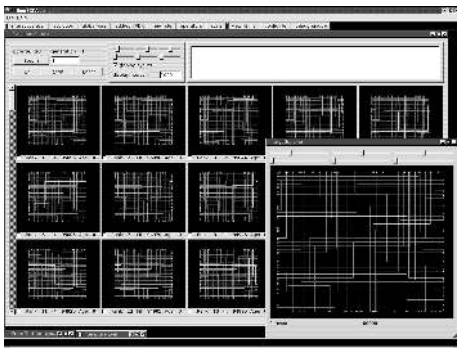

Fig. 3. Taken from an application of the HeurEAKA framework to the domain of switchbox routing: The GA window displays snapshots of the population at given intervals. The user can start, stop and step through evolution. An individual can be selected for closer inspection and evaluation, leading to Fig. 4.

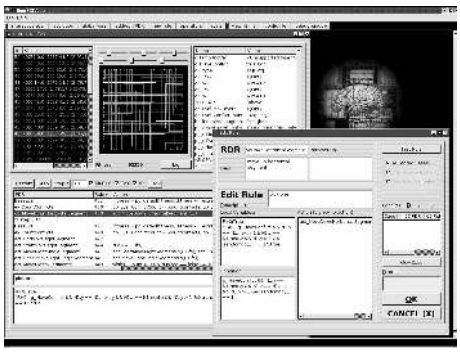

Fig. 4. The interactive screen allows the user to inspect the individual, step through rule application, and amend the KB as needed. More details in Fig.10.(Note that these screenshots were included to convey an overall impression only)

If the user does not agree with the performance of a rule R in the knowledge base, there are two ways of addressing it: Either the addition of an exception rule to R can be made, or a modification of R is possible.

Modification of rules in the KB is not normally done with Ripple Down Rules, as it is assumed that every rule which is ever entered into the knowledge base has its reason to be there. Also, a modification of a rule may have undesirable side effects which are not necessarily easy to control.

While these are valid reasons for not modifying rules, our initial experiments suggest that it is sensible to modify rules at the beginning of building a knowledge base. In particular, for the definition of new actions for modifying genomes, it proved useful to have this option to modify rules.

In traditional RDR, the system can suggest conditions for new rules based on past cases and the conditions of rules that did not fire. In our current framework this is not supported for two reasons. Firstly, the rule attribute space is very large

- it extends over expressions containing GA attributes, RSL variables and nested RDRs with parameters. Coming up with useful suggestions based on this large set is quite hard - one might consider machine learning methods for suggesting conditions. The second reason is that we are dealing with non-deterministic choices in the probabilistic algorithm. These choices are not reflected in the attributes tested by conditions, therefore past cases are not usually sufficient for determining how one arrived at a certain conclusion. Nonetheless, we hope that further work on methods for evaluating operator effectiveness might help in identifying useful rule construction strategies.

Indeed, it is part of our approach to be able to use a KB which is incomplete. i.e. Rules will not always be applied correctly since there is not always supervision by the user, also the selection of rules is left to chance, so there is no guarantee that an 'optimal' rule will be picked. Nonetheless, the heuristic search will be able to cope with an incomplete ruleset and still find useful solutions. The idea is that one provides generally useful operators and heuristic knowledge (e.g. in the form of operator weightings) and allow the genetic algorithm's search strategies and probabilistic selection to make up the rest of the algorithm.

User Interface: As mentioned previously, a good user interface is necessary for the productive development of rules. Visualisation of complex problems is a powerful way to elicit intuitive understanding from the user. Rules can be added easily and interactively, appearing in the same context as the visualisation. KB organization is handled transparently using RDR principles.

The user interface is re-usable for different problem domains. The only part that needs to be changed is the visualisation window. The GUI is implemented in Qt (a portable GUI toolkit by Trolltech), and the visualisation is wrapped in a single graphical widget object. This can readily be replaced by something that translates the genome into graphical format. Should the visualisation not be straightforward, requiring, e.g. interaction or animation, this can all be conveniently wrapped in this object. Along with the other modules in HeurEAKA, it should be relatively straightforward to apply to a new problem domain.

User Assistance: HeurEAKA provides some assistance to the user, helping to identify appropriate cases for modification of the KB:

A set of validation functions can be defined by the user, which are run on each individual during the execution of the GA. When a function identifies an appropriate individual, the GA is halted with an exception and provides a pointer to the individual along with a trace of all the rules applied in the last mutation operation. The user can then decide whether to further examine the individual. It is up to the user what he/she wants in the evaluation function, but we found it useful to include some general heuristic conditions which we found would indicate particularly bad characteristics, or even illegal solutions - allowing us to trace the origins of such individuals in the population.

Statistics are kept on rules involved in mutation operations. Changes in fitness are attributed to the rules that caused them, also the frequency of rule use in successful vs. unsuccessful individuals is tracked. These data help the user gain an overview of how useful particular parts of the KB are, and highlight problem areas that might need attention.

3 Case Study & Experiments

The following section provides an overview of two case studies done using HeurEAKA. A detailed description of the features available is not possible due to space limitation. We present how our framework was applied to the problem domain of channel and switchbox routing, along with experimental results.

3.1 Domain Specifics

In order to demonstrate that genetic algorithms enhanced with knowledge acquisition can be used to develop algorithms for solving complex combinatorial problems, detailed channel routing as well as switchbox routing, both industrially relevant problems within the realm of VLSI design, were chosen to demonstrate the approach.

A channel routing problem (CRP) is given by a channel of a certain width. On both sides of the channel are connection points. Each connection point belongs to a certain electrical net and all connection points of the same net need to be physically connected with each other by routing a wire through the channel and, of course, without two nets crossing. The width of the channel determines how many wires can run in parallel through the channel. The length of the channel determines how many connection points on both sides of the channel there may be. Furthermore, the layout is done on a small number of different layers (e.g. 2 to 4 layers), to make a connection of all nets without crossing possible at all. It is possible to have a connection between two adjacent layers at any point in the channel. Such a connection is also called a *via*.

The switchbox routing problem is similar to the CRP, but does not deal with only a two-sided channel, but rather a rectangle with connections on all sides. Since wires can originate and terminate on the sides of a switchbox, and the width of a channel generally being fixed (due to the fixed wire terminals on either side), the SRP is more difficult to solve.

A solution to the CRP and SRP will be referred to as a *layout*. A KB contains rules for the manipulation of layout instances, these can be used as part of a heuristic algorithm capable of solving layouts in the general case.

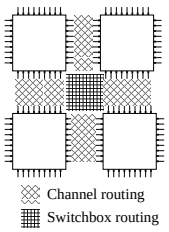

Fig. 5. Switchbox and channel routing in a general VLSI layout problem.

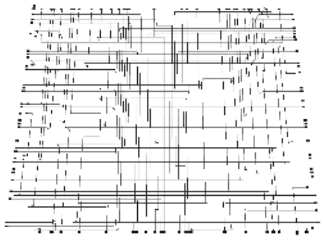

Fig. 6. An example of a switchbox solution containing 60 wires, 70 tracks and 100 columns.

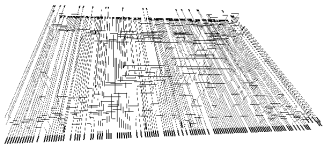

Fig. 7. A sample of a three layer channel routing layout with 150 columns (3 layer is easier than 2 layer).

Genome Encoding: A genome describes the layout of a CRP or SRP solution. This takes the form of a list of wires. The wires are numbered 0-w where w is the number of pin pairs that need to be connected, each connection being a wire. Each wire contains a sequential list of nodes. Each node has three coordinates corresponding to the row, column and layer number.

A layout is characterized by the number of columns and rows (tracks), how many pins (wire terminals) are to be found on the side of the channel / switchbox and what the pin configuration is. A problem instance used for the system would be given by the VLSI layout problem to be solved. Initially, a genome will usually not represent a valid solution as some wires are usually crossing. Only when all those crossings have been eliminated and not more than the prescribed number of layers are used would the fitness value of a genome reach a satisfactory level.

Genome Operations: Individuals in the GA are initialised with a random wire layout without regard to conflicts. The GA operates on a genome by crossover, mutation and evaluation.

The GA's crossover operation is currently not part of a knowledge base (Sect. 5 discusses this further). In crossover, two parents are duplicated giving two children. A set of wires is exchanged between the two children using 2 point crossover. This works by picking 2 random numbers, c_1 and c_2 (where $0 \leq c_1 \leq c_2 < number\ of\ wires$). The set of wires numbered c_1 to c_2 are exchanged between the children.

Evaluation is done using the evaluation KB. Typically the user uses as a fitness criteria the number of layers and conflicts in a layout. The length of wires, number of vias and possible cross-talk (electronic interference occurring in parallel wires) are also useful fitness criteria.

The mutation KB contains rules designed to manipulate the layout, typically they would describe the resolution of a conflict identified using the *.findconflict* command (a primitive function returning a conflict found in the layout).

Primitives Interface & Rule Specification Language Extensions: Primitives relating to the layout problem are supplied. These include the types *wire* and *node*, as well as other layout-specific commands. High level operators are defined as NRDRs forming a useful vocabulary for intuitive descriptions on the knowledge level. Pre-defined primitive functions are also supported, these include, for example, *.maxlayers* (counts the number of layers found in a layout), $.countconflicts$ (the number of conflicts found in a layout). Some describe aspects of the GA operation, for example $.ga.generation$ and $.ga.popmaxfitness$ (highest fitness in whole population).

Example of rules applied to the Switchbox Routing problem: Initially, a KB is built up defining operators using primitives based on node and wire manipulation. These form the foundation for more high-level concepts which can be used intuitively by an expert.

Assuming we start with a KB with relatively high-level actions defined, e.g. *RaiseWholeWire*, *MoveVerticalSegmentRight*, *MoveHorizontalSegmentDown* and *MoveHorizontalSegmentUp*, we can show how a sequence of these actions could be applied by the GA to solve a conflict, as seen in Fig. 8.

When applied to another example, Fig. 9 shows that the same sequence is unlikely to find a solution, and the expert can amend the KB to suggest an alternative action. A little background information might be in order: when improving a genome, the KB has rules which identify a random conflict to be fixed. This will tag the nodes immediately preceding or at the conflict of the two

conflicting wires - labeled here as *N1* and *N2*. 'Preceding' is defined in relation to the node ordering which starts at the bottom or left side of a channel/switchbox and ends at the terminating pin.

In this case, the user may find that *MoveVerticalSegmentRight* is undesirable, and formulate a rule with condition *is_Vertical(N1) && is_Horizontal(N2) && right_of(N2.next,N1)*, and action *MoveHorizontalSegmentUp(N1.prev)*. This rule would be added as an exception in the KB. The operators referenced here are defined as RDRs elsewhere in the KB, where *is_Vertical(N1)* returns true if the segment between *N1* and its succeeding node is vertical (change in row), *is_Horizontal(N2)* returns true if the segment between *N2* and its successor is horizontal (change in column). *right_of(N2.next,N1)* will return true if the node succeeding *N2* lies to the right of *N1*.

Figure 10 and Fig. 11 show how an expert would interact with the GUI to add the exception for the rule in Fig. 9.

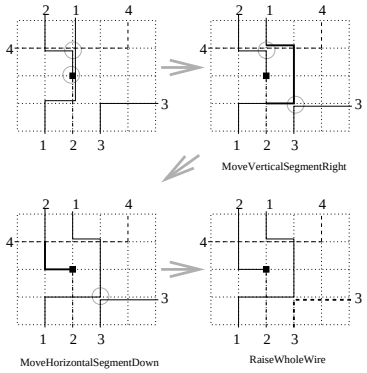

Fig. 8. Resolution of conflicts in a switchbox using 3 actions suggestions given by an expert. See Sect. 3.1. for a description.

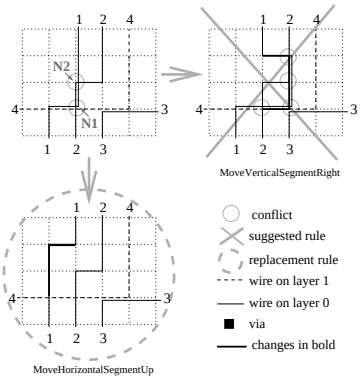

Fig. 9. An exception is created by the expert, replacing the first rule suggested by the KB, using the rule specified in Fig. 8. The tags N1 and N2 are explained in Sect.3.1

3.2 Channel Routing Experiments

In order to test our approach and the implemented tool, a KB was created. Initial tests were done with a KB containing 2 RDRs and 10 rules, later tests were run with 50 RDRs and 167 rules. On average the KA process took approximately 10 minutes per rule.

We tested HeurEAKA with a "minimal" KB in order to approximate a GA with very little domain knowledge, the 10 rule KB mentioned above was our "small" KB. Both of these KBs were relative trivial to create. With this we showed that without any useful knowledge about how to go about trying to solve a layout problem, the GA was unable to find a solution. The graph in Fig.12 shows how they performed. Initially, we see a reduction in conflicts due to the GA selecting those randomly initialised layouts with the least number of conflicts. The crossover operator accounted for some improvement too, since it spread useful sub-components of solutions through the population. Once the

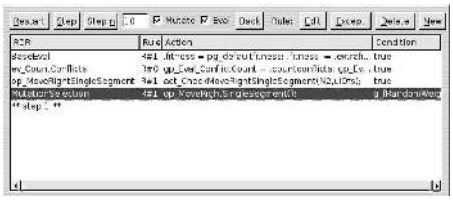

Fig. 10. The user is presented with a trace of rules selected for each mutation & evaluation step. When Nested RDRs are used, each RDR has a chosen rule.

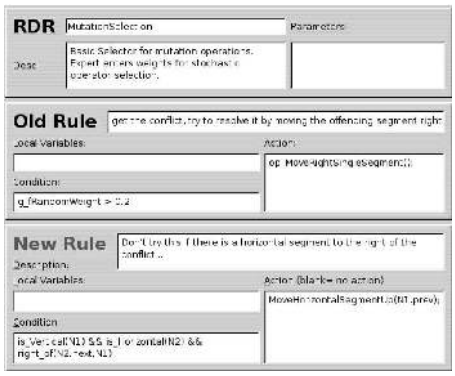

Fig. 11. An exception to the rule selected by the user in Fig.10 can be added.

easy part of the solution had been completed, both rule bases were ineffective for resolving the more difficult conflicts. The mature KB was able to solve the problems.

Layer Restriction: Initial tests using the 167 rule KB were run for 3 layer layouts, with some found for up to 150 pins a side, an example is shown in Fig. 7. We decided to concentrate on 2 layer solutions as they are more comparable to other attempts at solving the CRP, and the theoretical limits are better understood for 2 layer layouts.

Channel Width / Number of Tracks: For testing purposes, random pin configurations where generated. In order to approximate real problems and also to control layout *density*, a heuristic ratio of long distance to short distance crossing connections was used. This ratio was around 1:3.

The *density* for a 2-layer problem is defined as the maximum over all positions p along the length of the channel as follows: the number of nets which have at least one connection point on the left as well as one connection point on the right of p. This determines the theoretical lower limit on tracks needed to solve the problem.

CRP problems tested included 30,50,70 and 80 columns, with densities and track numbers of 20/25, 24/35, 39/50, 44/70 respectively. Our experiments show that reasonable solutions can be found using our approach. The size of layouts and track sizes look comparable to those benchmarks used with other CRP algorithms [11] and [7] - many are in the 12-23 column range, some up to 129.

3.3 Switchbox Routing Experiments

In order to see how flexible the HeurEAKA approach is, we tried to use the existing KB on a slightly different problem. We made modifications in the primitives module reflecting a different problem. We used a modified version of the KB developed for the CRP. Changing the primitives and adapting the KB from CRP to SRP took 2 days. Generally the SRP is more difficult to solve than

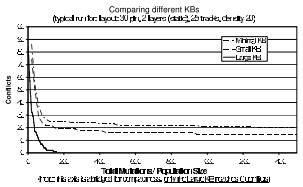

Fig. 12. Channel routing: Quasi random mutations "minimal" KB and a "small" KB are unable to solve the 30 pin layout problem. The more mature KB can solve it effectively.

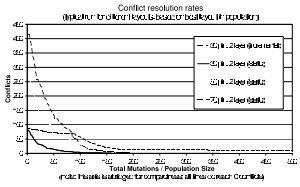

Fig. 13. Channel routing: Comparison of different conflict resolution rates, given different layout sizes and layer allocation strategies (static vs incremental).

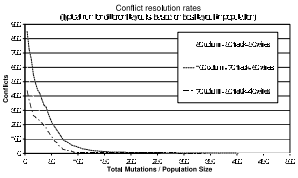

Fig. 14. Switchbox routing: Comparison of different conflict resolution rates, given different number of colums, tracks and wires.

the CRP. Track availability is far more restricted in the SRP since wire terminals are fixed on tracks (at the sides of the switchbox). Also, assumptions about terminals locations and wire orientation made in the CRP KB were not always valid for SRP. Nonetheless, it was found that due to the flexible nature of how HeurEAKA applies rules, it was possible to solve SRPs. In other words, in order to solve the CRP we encoded sufficient examples for solving problems that we generic enough to apply to the SRP. Also we are using a GA that is robust enough to recover from wrong operations and capable of functioning with incomplete operations.

Only two layer switchboxes were tested. Switchbox configurations were generated randomly with no restrictions on wire crossing distances. Solutions to switchbox layouts were found for (column number, track number, wire count): (30,20,20), (50,30,30), (70,50,40), (80,50,50), (100,70,60). Figure 14 shows conflict resolution rates for these switchboxes, and Fig. 6 shows an example of a switchbox solution.

The dimensions the SRPs solved compare favourably with the benchmarks tested in other attempts for do switchbox routing, notably when compared to those achieved by other GA approaches [8, 12].

4 Knowledge Acquisition Experience

A comprehensive KB was created, comprising 49 RDRs and 167 rules. Initial rules were low-level rules needed for the manipulation of nodes and wires. These were necessary for establishing sufficient operators to use in a knowledge level description. This work was fairly technical and required repeated editing and debugging to ensure operator correctness. Incorrect operators at the low level would often create invalid layouts (diagonal wires or vias, moved wire terminals) or undesirable wiring - double-backing, repeated nodes etc. In order to help the user, a validation module was run during a genome's evolution, and any invalid layouts would generate an exception. The offender could then be hand-analysed (the exception generates an execution trace).

After the low level rules were defined, the KA process became easier since most rules could be defined at a higher level of abstraction. Because this was a more intuitive abstraction level, it was easier to formulate rules and required

less revision of conditions or actions. This part of the KB is primarily what the probabilistic algorithm makes use of: choosing from the different high level strategies in an attempt to resolve conflicts.

Using some of the simple rule statistics provided by HeurEAKA, it was possible to identify some dud rules, i.e. those that attracted excessive fitness penalties. In general it was felt that these statistics needed to be further developed to be really useful.

Another general strategy used for formulating rules was to let the GA run and wait for the population to converge to promising solutions. Once the population fitness rates had plateau'd, it was assumed current rules could suggest no better improvements. Individuals were pulled out of the population and rules added where the expert could suggest a strategy for improving on them.

5 Discussion

The experiments show that it is feasible to attempt solutions to complex problems using the HeurEAKA framework - in this case we applied it to the domain of detailed Channel Routing Problem (CRP) and the Switchbox Routing Problem (SRP). We were able to effectively perform KA using ripple-down rules based knowledge acquisition, which formed an effective KB for the GA.

The development of the problem specific components of HeurEAKA did require some effort in addition to KB construction, it would be commensurate with any other attempt at solving a CRP/SRP, since basic encoding and access of a layout is necessary. We argue that the additional effort usually spent on adapting and tuning GAs towards a problem domain is in excess of ours.

A number of studies have been made in the application of GAs to CRP and SRP, these make extensive use of domain knowledge by formulating either custom operators, or using existing known routing techniques [7, 8]. [7] show that the use of informed operators results in finding of better solutions for the CRP, which is what one would expect.

It is known that the development of a suitable representation, mutation and crossover operators for genetic algorithms is often quite difficult [1]. Instead of relying on the definition of operators through expert introspection and trial and error, we allow the user to formulate them by exploring example cases. There is an approach somewhat related in the use of case based reasoning for GA in VLSI design [9]. Here, however, previous cases are selected by an expert and only used for injection into a GA search, rather than formulation of operators. It does not build on generalizations and expert insight learned from these cases, thus being far less powerful.

There is some controversy over the importance of mutation vs. crossover operators [2] in the successful design of GAs. In our current implementation, we have chosen to initially concentrate on KA for the mutation and evaluation operators.

It is worth investigating the use of KA for replacing the current crossover operator with a KB based one. Goldberg [2] explains that in selectorecombinative GAs, the design of crossover operators is very important for successful search. When choosing the operator one wants a good chance of preserving sub-solutions when exchanging between two individuals. When designing a crossover operator, initially a user is likely to have very little prior knowledge in matching the crossover operator to the problem encoding [2]. With KA one can support the

iterative refinement of the crossover operator in a similar way to our existing approach. We expect, however, that it will not be as easy for a user to create rules identifying good strategies for sub-component exchange as it is for incremental improvement of one genome (as is currently done for the mutation operator).

We have provided the user with useful tools in the formulation of a KB. The KA process was effective and supported by a good user interface. Helper functions providing some automated selection of cases which would be good candidates for formulation of new rules have been attempted. Extensions could include the ability to review new and exisiting rules against a case history, and better statistical measures relating their use in successful and unsuccessful evolutionary paths.

On average rules took approximately 10 minutes each to formulate, taking about 30 hours for the formulation of a viable knowledge base. The formulation of effective CRP and SRP algorithms has been the subject of much study and industry-standard algorithms took many years to develop [10]. In our case KA was done by a novice, using mainly intuition and being able to incrementally specify rules in a natural way on the knowledge level. Thus the effort and expertise required was significantly less than commercial routing solutions.

Direct comparison of our solution to existing benchmarks needs extension to supporting wire nets in the HeurEAKA tool. The results outlined in the previous sections look promising, and with the continued development of the KB, should produce even better results in the future.

The application of our framework in the well understood domain of CRP and SRP, enables us to benchmark our results against industrially used algorithms.

6 Conclusion

In this paper we have presented a framework for solving complex combinatorial problems based on incremental knowledge acquisition from a human expert. We outline how our approach makes it easier to tackle such problems than the conventional design of algorithms.

Given that the development of standard genetic algorithms still requires considerable effort in the formulation and tuning of operators, we introduce a method that is better suited to integrate domain knowledge. We used NRDR, an unconventional KA technique, by integrating it into the design process to provide an intuitive method of supporting an expert's development effort.

We have adapted the principles of RDR to apply them in probabilistic search algorithms. Considering the use of unusual RDR characteristics, our approach showed that they still allowed us to perform effectively.

We demonstrate our approach in the domain of detailed channel and switchbox routing. This shows that the approach achieves results comparable to conventional approaches developed with considerably more effort.

We also hope to extend the RDR techniques used to provide more automated support for rule formulation such as the automatic evaluation of proposed new rules on databases of genomes that were generated through previous genetic searches. Currently the process of identifying good candidate solutions for rule formulation requires a fair amount of interaction by the user. In future this process can be more automated by leveraging performance statistics of mutation operators, as well as the possible introduction of some machine learning techniques.

The knowledge base used in the experiments for CRP and SRP is sufficient to solve problems comparable to those used in other GA benchmarks. However, in future work we plan to extend our implementation to be able to tackle more challenging problems. If we can show the competitiveness of these solutions, we hope to apply the framework to problems in other domains.

References

1. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Springer Verlag (2002)
2. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Volume 7, Kluwer Series on Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (2002)
3. De Jong, K., Spears, W.: Using genetic algorithm to solve NP-complete problems. In Schaffer, J.D., ed.: Proc. of the Third Int. Conf. on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann (1989) 124–132
4. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press. (1975)
5. Compton, P., Jansen, R.: Knowledge in context: A strategy for expert system maintenance. In: 2nd Australian Joint Artificial Intelligence Conference. Volume 1. (1989) 292–306
6. Beydoun, G., Hoffmann, A.: Theoretical basis for hierarchical incremental knowledge acquisition. In: International Journal in Human-Computer Studies. (2001) 407–452
7. Gockel, N., Pudelko, G., Drechsler, R., Becker, B.: A hybrid genetic algorithm for the channel routing problem. In: International Symposium on Circuits and Systems, volume IV. (1996) 675–678
8. Lin, Y., Hsu, Y., Tsai, F.: Silk: A simulated evolution router. In: IEEE Transactions on CAD. Volume 8.10. (1989) 1108–1114
9. Liu, X.: Combining genetic algorithm and casebased reasoning for structure design (1996)
10. Lengauer, T.: Combinational Algorithms for Integrated Circuit Layout. B.G. Teubner/John Wiley & Sons (1990)
11. Lienig, J., Thulasiraman, K.: A new genetic algorithm for the channel routing problem. In: 7th International Conference on VLSI Design, Calcutta (1994) 133–136
12. Lienig, J.: Channel and switchbox routing with minimized crosstalk - a parallel genetic algorithm approach. In: the 10th International Conference on VLSI Design,Hyderabad (1997) 27–31