Improved Knowledge Acquisition for High-Performance Heuristic Search

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.
{jbekmann,achim}@cse.unsw.edu.au

Abstract

We present a new incremental knowledge acquisition approach that incrementally improves the performance of a probabilistic search algorithm. The approach addresses the known difficulty of tuning probabilistic search algorithms, such as genetic algorithms or simulated annealing, for a given search problem by the introduction of domain knowledge. We show that our approach is effective for developing heuristic algorithms for difficult combinatorial problems by solving benchmarks from the industrially relevant domain of VLSI detailed routing.

In this paper we present advanced techniques for improving our knowledge acquisition approach. We also present a novel method that uses domain knowledge for the prioritisation of mutation operators, increasing the GA's efficiency noticeably.

1 Introduction

Searching for good or even optimal solutions for complex combinatorial problems has been a challenge in Artificial Intelligence research ever since AI's earliest days.

While it has been long recognized that the effectiveness of a particular search approach depends on the chosen representation as well as certain characteristics of the problem to be solved, to date there is no systematic approach available that allows one to efficiently engineer a special-purpose search strategy for a given search problem.

In this paper we present an approach that allows humans to articulate their intuition about how to search effectively for a given search problem. This appears to be a promising approach as humans often have good intuitions about how to find acceptable or good solutions to search problems, if they had to search manually.

We embedded our approach in the probabilistic search framework of Genetic Algorithms (GA). While GAs have a reputation as being general purpose, in practice it requires usually substantial adaptation effort to the search problem at hand. For complex problems this process may well take months in order to allow the GA to find an acceptable solution within reasonable time.

We chose a probabilistic search algorithm in which to embed the problem specific knowledge about effective search strategies as this allows one to stop the knowledge acquisition process at any time. A smaller knowledge base (KB) results in a less effective search process, but may still find a satisfactory or near satisfactory solution for less challenging problem instances. Incremental additions to the KB improve the effectiveness of the search.

As part of our discussion, we present a novel technique of varying GA mutation strategy based on execution history, having implications beyond our work for GAs in general.

For the knowledge acquisition process we build on the idea of Ripple Down Rules [3], which allows us to incrementally refine the given (initially empty) knowledge base. Ripple Down Rules (RDR) ensure that a knowledge base is only amended in a way that does not deteriorate its accuracy on previously seen instances.

Our variant of RDR is a significant extension of the RDR approach. We also introduce an expanded knowledge acquisition process in which we help the expert find examples in past search history where the quality of the KB needs attention. This work is based on [1].

In section 2 we provide a short overview of genetic algorithms and RDR. This is followed by section 3 where we describe our framework HeurEAKA!. Section 4 introduces a case study of how HeurEAKA! was applied to detailed routing, an industrially relevant problem in VLSI design. We discuss our experiments and results to these in section 5, demonstrating that our approach is successful. The conclusions follow in section 6.

2 Background

Before we discuss our approach to combining GAs and Knowledge Acquisition, we start with a brief summary of these two areas:

2.1 Probabilistic Search with Genetic Algorithms

A number of evolutionary algorithms exist; we base our approach on genetic algorithms which is one such algorithm. Basic GAs are relatively easy to implement. A solution candidate for the given problem instance is encoded into a genome. A set of genomes makes up a population of potential solutions. The GA performs a search through the solution space by modifying a population of genomes, guided by an evolutionary heuristic. When a suitable solution has been identified, the search terminates.

GAs usually start with a randomly initialized population of individuals, and guide their search by a *fitness* function of the individuals. In a probabilistic fashion, using operators for selection, mutation and crossover, the GA attempts to direct the search to promising areas. I.e. the GA maintains a certain population size by replacing less fit individuals by newly generated ones. The new individuals are generated by mutation or cross-over based on parent individuals of high fitness.

Due to their generality and flexibility, GAs have been applied in many domains, demonstrating their effectiveness on hard problems [4; 6]. In very complex domains there are significant challenges where generic GA techniques do not scale well. This includes high sensitivity to problem encoding and operator formulation, selection strategies, as well as selection of operator weightings, population size and running time[7; 13; 6]. Thus in practice substantial tuning of the GA is necessary to augment the generic algorithms.

In our approach, we do not use an encoding of our problem into a binary code as is often seen with conventional GAs. Instead, our encoding is a direct representation of a solution (see below for a better description). Our crossover operator is designed for this representation and thus is very effective in maintaining sub-solution coherence in the genome, and also identifying which areas of the genome appear fitter than others. We have a simple yet effective crossover operator, and chose to concentrate our work on the mutation operator - mainly because it was better suited to our the knowledge acquisition approach.

Our approach integrates with the GA a knowledge base where humans can codify their intuition about useful search steps and strategies for the given type of problem.

In addition, we present a novel method that uses domain knowledge for the prioritisation of mutation operators, increasing the GA's efficiency noticeably. Because we have sufficient domain knowledge in our fitness function, we can identify parts of the genome that are desirable or undesirable and use that knowledge to influence mutation strategy. We also show how using the mutation history of these identified characteristics to change mutation selection strategy.

2.2 Knowledge Acquisition using Ripple Down Rules

RDR [3] allows one to add rules to a knowledge base incrementally without compromising the previously shown satisfactory performance of the KB. An extension of RDR allows hierarchical structuring of RDRs - "nested RDR" (NRDR) [2]. NRDR allows the re-use of definitions in a KB, as well as the abstraction of concepts, which allows for more compact KBs.

Single Classification Ripple Down Rules (SCRDR) use 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.

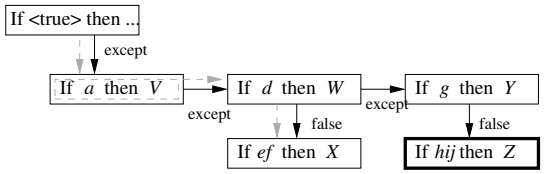

Figure 1: A simple RDR structure. The dashed line indicates the path taken & rule executed (V) for $a\overline{de}$, the bold box indicates how action Z would be added for $ad\overline{g}hij$.

Figure 2.2 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.

For the purpose of integrating with the Genetic Algorithm, in our approach each case is a genome and conclusions are actually a sequence of actions that can be applied to the genome.

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 part of an action, see section 3). 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.

3 HeurEAKA! - Our Framework for Improving Search with Domain Knowledge

The HeurEAKA! (Heuristic Evolutionary Algorithm with Knowledge Acquisition) framework is composed of a general GA scheme and a knowledge acquisition (KA) component.

Each genome in the GA population represents a potential solution to the problem to be solved. While searching, the GA needs to perform a fitness evaluation and the mutation of a genome. For this it uses the two knowledge bases, the *Mutation KB* and *Fitness KB*. The GA presents a genome to the relevant KB, which responds by suggesting a mutation or fitness score. (The crossover function is handled differently, more details below).

The KA module handles maintenance of the KBs, which contain collections of NRDRs. The NRDRs are incrementally developed by the expert through the addition of rules

(the next section contains a detailed description of this process).

The *Fitness KB* contains rules based on past examples given by the expert about desirable and undesirable traits of genomes. The GA requires a fitness value, so the rules contain various calculations the expert adds to reflect these traits.

The *Mutation KB* similarly contains rules accumulated from expert recommendations. Each rule contains an action to be executed under certain conditions. Each action is one or more commands which can modify the genome. The KB contains a set of NRDRs which contain these rules. When a genome is presented to the Mutation KB, it is evaluated against these NRDRs. In accordance to the RDR algorithm, actions which satisfy the relevant conditions are selected for execution. A non-deterministic element is introduced by letting the expert attach probabilistic weights to selected NRDRs and letting a random procedure pick which one to execute. This allows the expert leeway to speculate about what the best actions might be, effectively letting the GAs heuristics "decide" the most appropriate action.

As opposed to other methods related to case-based reasoning and GAs (e.g. [12] or [11]), the expert uses past cases to formulate rules to explain improvements, rather than using past examples or similar solutions injected into the evolutionary process, or trying to extract the rules automatically.

The overall architecture of HeurEAKA! is domain neutral. The implementation contains KB management modules, GA, a graphical user interface as well as range of utilities supporting the expert evaluate past searches. This is complemented with a smaller component that needs to be implemented for each problem domain containing a relevant solution representation. The user interface also requires a small addition to translate this representation for visualisation.

A general purpose language is extended with primitives to make it suitable for describing the domain specific representation. Rules entered by the expert are specified using this simple language, which is loosely based on "C" syntax. The rule conditions use logical expressions while actions include a range of commands, including variables, loops and references to other NRDRs which are evaluated analogously to function calls. Section 5 gives examples of rule specification.

3.1 The Knowledge Acquisition process in HeurEAKA!

The overall approach allows the user to run the Genetic Algorithm with the current KBs and monitor or inspect afterwards whether the fitness function computed for an individual as well as the applied mutation operator where appropriate/promising steps to do.

To do that, the genetic algorithm can be started, stopped and reset via a graphical user interface. A snapshot of the GA population is presented visually, from which the user can pick an individual for closer inspection.

Figure 2 shows an individual genome being inspected. A user can step through the process of mutation and evaluation of that genome as if the solution was part of the GA population. Given non-deterministic elements of operator selection, the interactive debugger maintains complete state descriptions, allowing the user to step forward and backwards in execution history and also recreating conditions for repeated testing.

The expert can review actions applied to the genome and make modifications to the KB if he/she feels they are needed. The modifications are typically done by adding exceptions to the NRDR (in accordance with RDR methodology described above), but may also be an edit of an existing rule or the addition of a new NRDR.

In the traditional Ripple Down Rules approach it is assumed every rule entered is correct and has its reason to be there, modification of a rule may have undesirable side effects which are often difficult to control. While these are valid reasons for not modifying rules, our experiments suggest that it is sensible to modify rules at the beginning of building a knowledge base. In particular, for the definition of mutation operators it proved useful to have this option to modify rules.

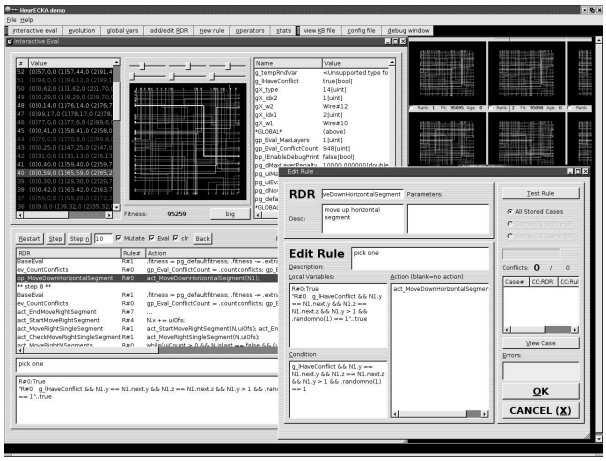

Figure 2: The interactive screen allows the user to inspect the individual genome, step through rule application, and amend the KB as needed.

3.2 Advanced features for supporting Knowledge Acquisition

Once a sufficiently large KB has been created based on small-scale evaluation, more automated methods are needed to support a user's interaction:

Genome Visualisation: One of the strengths of using RDR is that the approach is successful in eliciting tacit knowledge [3], since rules are formulated while the system is in use and with the justification of an example. With complex problems (such as is described in section 4), the visualisation of a solution is very important in allowing the user to form an intuitive understanding. In the evaluation module, there is immidiate visual feedback when rules are applied.

Trigger functions: During the execution of the GA, thousands or even hundreds of thousands of rule applications to all manner of genomes are made. It is not possible for the expert to supervise each case. The user can define a trigger function which is applied to each genome during the execution of the GA. If the function's condition is met, an exception is

thrown, halting the GA's execution, providing a pointer to the individual along with an execution trace of recently applied rules.

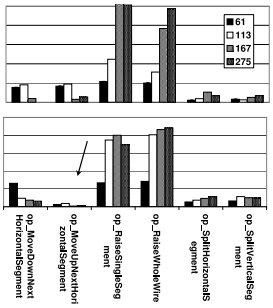

Figure 3: NRDR usage in genome. The legend shows the number of conflicts (less is better). Arrow: an NRDR which is much less useful after an error was introduced by the expert (the top graph is before the error, the lower one after).

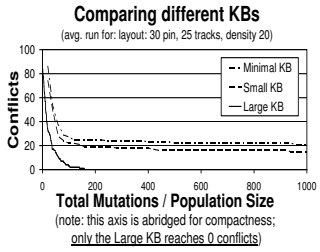

Figure 4: The "minimal" KB and "small" KB are unable to solve problem. The more mature KB can solve it effectively.

NRDR usage profile: Execution statistics are kept for each NRDR execution, recording the frequency of use over many GA runs. Comparing the profile of aggregate use of NRDRs by successful vs less successful genomes (judging by their fitness function and how soon they were eliminated from the GA population), provides some impression of an NRDRs "usefulness" and role in GAs lifetime. Figure 3 shows the frequency of a few selected NRDRs relative to the genome's age (the examples are from the VLSI case study discussed in the next section).

In the less fit genomes, some operators have a high count compared to the fitter genomes. These operators turn out to be "high churn" operators - they can affect large sections of a genome. In the earlier stages of a GA these NRDRs support high entropy and thus allow the search to cover more search space. When the genome is closer to the ideal solution, i.e. has more desirable characteristics in place, these NRDRs disrupt established "good" configurations. Their relative usage frequency in fitter populations is thus lower.

The characterisation of NRDR utility can only provide the expert with a rough idea of its "usefulness". In that context it did contribute somewhat to the intuition the expert developed when doing KA. Where it was found to be quite useful though, was when the expert introduced some modification to the NRDR which inhibited it from functioning as well as previously - meaning that it was much less used in arriving at fit genomes. This helped catch at least a few of errors. In figure 3 the arrow indicates such an NRDR.

Tuning probabilistic weighting of NRDRs: In section 3 we mentioned the notion of using probabilistic weightings to affect the GA's selection of NRDRs. This allows the expert to "tune" the KB when he/she has a good intuition as to what action might be useful, but is not entirely sure.

In our experiments we identified a set of NRDRs which would probably be useful for various types of problems. For these we created weightings giving preference to the relevant NRDRs. This was in the final stages of refining our KB, where we allowed 1/4 of all modifications to be decided randomly, and 3/4 based on the probabilistic weightings based on problem classifications. With this we saw a measurable improvement in our search results.

In the experiments conducted in our test domain, we saw a measurable improvement in our search.

Identifying genome modifications of interest: Sometimes a genome mutation plays a role in the long term success of that genome, but the immediate result of the mutation is a reduction or no change in its fitness. In general, GAs cope with this by having a large enough population pool and preventing premature fitness convergence, so as not to throw out such genomes too soon. In traditional GAs it is hard to otherwise account for these cases. Given that we have an expert who can make judgment on individual examples using intuition and hindsight, the fitness function can be augmented to bridge the interval needed to see a promising mutation come to fruition.

The problem comes with identifying these candidate mutations in order to present them to the expert. We identified all the mutations falling into a window size of 10 actions prior to a fitness improvement exceeding a given threshold. These were collected for review for the expert's consideration. In practice we found there was too much data to effectively judge the cases by hand.

We then tried using a discounted reward scheme, and some other methods to help reduce the candidate space, but have not yet been able to improve matters. This aspect of the project are currently being worked on.

3.3 Advanced features for use in Knowledge Acquisition

Preferences in choosing parts of the genome to mutate: As hinted at in the genetic algorithms section (2.1), we have introduced a novel method of choosing parts of a genome for mutation. In our fitness function we are able to identify parts of the genome that are undesirable.

We keep track of the modification of those parts over time, and give a higher preference weighting to newly arising problem areas. This is done by keeping a list of existing problem areas and discounting their preference weighting after each fitness evaluation. When it comes to choosing a part of the genome to mutate, we normalise these weights and make a probabilistic choice of the identified problems. This results in newer problem areas having a higher chance of being picked. With this scheme we allow some continuity in search direction, potentially overcoming local minima in the search space.

In the context of our experimental domain, we empirically found a 65% discount factor gave the best results. Compared with no discount scheme (giving all identified problem areas equal chance) the number of successful GA trials was 85% higher, and the successful trials took on average half the time to complete.

Mutation look-ahead: We added a feature that allows the expert to make use of a "mutation look-ahead" function in

the rule specification. This function allows for the comparison of a genome before and after a mutation has been made. This effectively allows a local search in the mutation NRDR (similar to a Lamarckian learning mechanism in evolutionary algorithms).

Our knowledge base incorporated local search by comparing the fitness of a genome before and after a candidate mutation. This can be used in an RDR at any point, but was most successfully used for a generalised mutation look-ahead. Here, for each mutation a series of 10 steps, each recommended by the mutation KB was used, while the fitness values were recorded. The point of highest fitness value in the sequence was picked as the final mutation.

When using the look-ahead mechanism, GA trials had a significantly higher success rate. Each successful trial was also faster to converge on a solution (as measured in overall execution time).

4 Case Study - Detailed VLSI Routing

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.

4.1 Domain Specifics

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. Two adjacent layers can be connected at any point using a so-called *via*.

The switchbox routing problem (SRP) is similar to the CRP but usually more difficult, as it deals with connections on all four sides rather than only two. A solution to the CRP and SRP will be referred to as a *layout*.

Genome Encoding: A genome describes the layout of a CRP or SRP solution. This takes the form of a list of straight wire segments.

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. However, the GA's crossover operation is currently not controlled by a knowledge base. The crossover for our layout problems groups wires into mutually exclusive sets, one of those sets is exchanged between parents (i.e. the corresponding wires are exchanged). The crossover operator is thus highly sensitive to the structure of the problem, as well as interaction between its elements. The formulation of the crossover operator is domain specific, and we plan to make it part of the knowledge acquisition cycle in future versions.

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).

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. *MoveVerticalSegmentRight*, and *MoveHorizontalSegmentUp*. When applied to the example in figure 5, it seems that the action *MoveVerticalSegmentRight* is unlikely to lead to a promising solution. The expert can amend the KB to suggest an alternative action. (see nodes labeled *N1* and *N2* in figure 5). In this case, the user will 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. *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*.

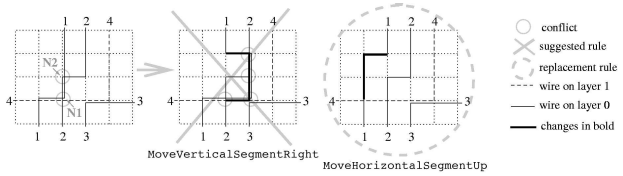

Figure 5: An exception is created by the expert, replacing the action *MoveVerticalSegmentRight* suggested by the KB, with the action *MoveHorizontalSegmentUp(N1.prev)*. (The tags N1 and N2 are referenced in the text).

5 Experiments and Results

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 and 53 RDRs and 182 rules.

In order to show that the introduction of domain knowledge improved the search, we tested three different KBs: A "minimal" KB with 5 rules, a "small" KB containing 10 rules, and a "large" KB containing 167 rules. Using the first two, the GA was unable to solve the given problem, while using the third, it was. Fig.4 shows progressive improvement in search with KB size.

A KB containing 53 RDRs and 182 rules was created. Initial rules were low-level in nature, dealing with manipulation of nodes and wires. This technical work required frequent editing and debugging. Using the validation module with trigger functions (described above) was helpful at this stage.

After the low level rules were defined, 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. The high level NRDRs were used by the mutation KB, for what the probabilistic algorithm was intended: choosing from the different high level strategies in an attempt to resolve conflicts.

Results: We were able to solve well recognised benchmarks from the domain of switchbox and channel routing. These included Burstein's difficult channel routing problem, Burstein's switchbox routing problem, the DENSE switchbox and the JOO_6_16 problem, amongst others. The solutions found were comparable to others' attempts, including those of the WEAVER, Silk, Packer and Monreale routers [9], [5], [10].

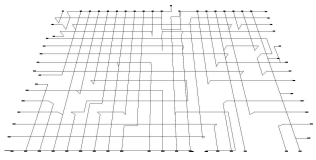

Figure 6: Solution to Burstein's switchbox problem.

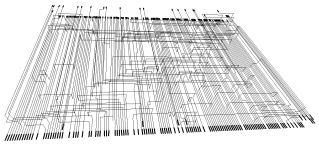

Figure 7: A sample channel routing solution.

We found it initially difficult to complete some of the more challenging benchmarks until we used the techniques listed in sections 3.2 and 3.3

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 [8]. 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.

6 Conclusion

We have presented an approach to solving complex combinatorial problems by using a generic search algorithm and incrementally introducing domain knowledge in order to make the search tractable. We use knowledge acquisition that supports the exploratory development of solutions, which, when combined with a GA, makes it easier to tackle difficult problems than having to design a conventional algorithm.

While GAs have been successfully used in many domains, in practice they often require considerable augmentation to make the generic architecture effective in complex domains. The unconventional KA technique of RDR formalises this adaptation process, allowing an expert to intuitively develop a domain-specific solution. In order to use RDR for integration with GAs, we have introduced new methods to make them work with probabilistic search.

We have also presented a number of advanced techniques that enhance the KA process. These include the use of trigger functions, NRDR usage profiles and weightings, mutation look-aheads and automatic case identification.

As part of our mutation operators, we have identified a promising technique for improving probabilistic search by using the mutation history of a genome to bias the search into more recent areas of exploration. We have found this heuristic to return promising results.

The application of our framework in the well understood domain of detailed channel routing and switchbox routing, enables us to compare our results against industrially used algorithms. We have shown that we are able to solve accepted benchmarks competitively, using far less effort than needed for the development of conventional algorithms.

In future work we will investigate the application of our techniques to a different domain. Another worthwhile direction would be to explore the applicability of our advanced KA techniques to more traditional KA domains outside of search.

References

[1] Bekmann, J.P., Hoffmann, A.: Incremental Knowledge Acquisition for Improving Probabilistic Search Algorithms. In: 14th International Conference on Knowledge Engineering and Knowledge Management (2004) 248–264.

[2] Beydoun, G., Hoffmann, A.: Theoretical basis for hierarchical incremental knowledge acquisition. In: International Journal in Human-Computer Studies. (2001) 407–452

[3] 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

[4] 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

[5] 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

[6] 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)

[7] Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press. (1975)

[8] Lengauer, T.: Combinational Algorithms for Integrated Circuit Layout. B.G. Teubner/John Wiley & Sons (1990)

[9] Lienig, J., Thulasiraman, K.: A new genetic algorithm for the channel routing problem. In: 7th International Conference on VLSI Design, Calcutta (1994) 133–136

[10] Lin, Y., Hsu, Y., Tsai, F.: Silk: A simulated evolution router. In: IEEE Transactions on CAD. Volume 8.10. (1989) 1108–1114

[11] Liu, X.: Combining genetic algorithm and case-based reasoning for structure design. Masters Thesis (1996), Department of Computer Science, Univ. of Nevada.

[12] Perez, E.I., Coello, C., Aguirre, A.H.: Extraction and reuse of design patterns from genetic algorithms using case-based reasoning. In: Soft Computing 9(1) (2005) Springer 44–53.

[13] Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Springer Verlag (2002)