Automating the Generation of VNS Components with Grammatical Evolution

Ender Özcan, John Drake and Nikolaos Kililis

School of Computer Science, University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{Ender.Ozcan, psxjd2, nxk09u}@nottingham.ac.uk

Abstract. The vehicle routing problem (VRP) is a family of problems whereby a fleet of vehicles must service the commodity demands of a set of geographically scattered customers from one or more depots, subject to a number of constraints. Early hyper-heuristic research focussed on selecting and applying a low-level heuristic at a given stage of an optimisation process. Recent trends have led to a number of approaches being developed to automatically generate heuristics for a number of combinatorial optimisation problems. Previous work on the VRP has shown that the application of hyper-heuristic approaches can yield successful results. In this paper we investigate the potential of grammatical evolution as a method to evolve the components of a variable neighbourhood search (VNS) framework. In particular two components are generated; constructive heuristics to create initial solutions and neighbourhood move operators to change the state of a given solution. The proposed method is tested on standard benchmark instances of two common VRP variants.

1 Introduction

Optimisation problems often create a search space which is too large to enumerate and exhaustively search for an optimal solution. Various heuristics and metaheuristics have been successfully applied to such problems. One drawback of such approaches is the necessity to manually adapt the method used to solve different problem domains or classes of problem. Hyper-heuristics are a class of high-level search techniques which automate the heuristic design process and aim to raise the level of generality at which search methods operate [1]. Hyper-heuristics are broadly split into two main categories, those which select a low-level heuristic to apply at a given point in a search and those which create new heuristics from a set of low level components [2]. Here we are concerned with the second category, those methodologies which generate new heuristics. Whilst most research effort in this field has been on developing heuristics which construct a solution from scratch, a less studied area is the generation of perturbative or local search heuristics. Genetic Programming (GP) [3] has been widely used in the literature to generate heuristics for strip packing [4], bin packing [5–7], job shop scheduling [8], knapsack problems [9] and boolean satisfiability [10–13]. Grammatical

Evolution [14] is a grammar-based variation of GP which has been used to automatically design local search heuristics bin packing and stock cutting problems. The vehicle routing problem (VRP) is an NP-Complete [15] combinatorial optimisation problem, which requires the determination of the optimal set of routes to be followed by a fleet of vehicles in order to service the commodity demands of a set of customers. Previously, hyper-heuristic methods [16] have shown to perform particularly well on a number of variants of the VRP. In this paper we explore the potential of grammatical evolution as a method to generate both constructive and perturbative heuristics for the VRP and embed these ideas in a variable neighbourhood search (VNS) framework.

2 Hyper-heuristics

The underlying principles of hyper-heuristics were used as early as the 1960's in the work of Fisher and Thompson [17], where combining job-shop scheduling rules by selecting an appropriate rule for the given state of a problem was shown to outperform using each of the rules individually. The term was first used in the field of combinatorial optimisation by Cowling et al. [18] defining hyper-heuristics as *heuristics to choose heuristics*. Unlike traditional computational search methods which operate directly on a search space of solutions, hyper-heuristics operate exclusively on a search space of heuristics or heuristic components. Burke et al. [2, 19] define a hyper-heuristic as *a search method or learning mechanism for selecting or generating heuristics to solve computational search problem*. This definition distinguishes between the two main classes of hyper-heuristics, those which intelligently select a heuristic to apply to a problem and those which are concerned with automatically generating new heuristics.

The automated generation of heuristics is a relatively new field attracting an increasing amount of attention. Genetic Programming (GP) has been successfully used to evolve heuristics for a wide range of problems. In genetic programming, populations of computer programs are evolved using the naturally inspired notions of inheritance, selection and variation. Unlike Genetic Algorithms which produce fixed-length encoded representations of candidate solutions to a given problem, the evolved program itself when executed is the solution. Geiger et al. [8] used GP to evolve dispatching rules for a job shop scheduling problem. Burke et al. create heuristics for for strip packing [4] and bin packing problems [5–7] with human-competitive results. Bader-El-Din and Poli [10] also used GP to quickly generate 'disposable' heuristics for the satisfiability problem generating heuristics again with comparable performance to human-designed methods. Fukunaga [11–13] also used GP to generate local search heuristics for boolean satisfiability. Drake et al. [20] managed a constructive heuristic by using GP to evolve the order in which to add items to a knapsack solving the MKP. Further information on using genetic programming as a hyper-heuristic is provided by Burke et al. [21].

3 Grammatical Evolution

Grammatical Evolution (GE) [14] is a recently developed grammar-based form of genetic programming. The evolutionary process in a grammatical evolution system is performed on binary or decimal integer strings of variable length rather than on actual programs. Such strings are then mapped to a sentence (in our case a program) using the production rules of a grammar expressed in BNF (Backus Naur Form). Unlike GP, GE provides a distinction between the genotype and phenotype as is the case in nature. In GE the search process is performed over the genotype (a binary or decimal integer string) and the fitness function evaluates the program (the phenotype) which is obtained. There are a number of advantages to approaching the search process in this way. Any strategy that operates on binary strings can be used to perform the search, this is not strictly limited to evolutionary approaches. The search is also not limited by tree structure and the need to ensure solutions are valid. Within the BNF notation a possible production rule can be defined as:

<symbolA> ::= <symbolB> | (symbolC)

In the example above, <symbolA> is a non-terminal which expands to either <symbolB> or (symbolC). <symbolB> is also a non-terminal, while (symbolc) is a terminal symbol indicated by brackets. The process typically starts with a single non-terminal start symbol and a set of production rules that define with which symbols this non-terminal can be replaced. A sentence can consist of any number of non-terminal and terminal symbols. Terminals are atomic components of a sentence containing no production rules as they will not be replaced. Each non-terminal is replaced with any non-terminal symbols produced subsequently replaced using their own corresponding production rules. Often there are multiple production rules to replace the current non-terminal symbol in question and a number of choices for terminal symbols. In order to select a symbol at a given point the variable length binary or decimal integer string representing the genotype in the GE system is used. In the case of a binary string, the genome is split into 8-bit portions known as codons. The integer value of a single codon can then take any value between 0 and 255. If the genotype is represented directly as a decimal integer string then this conversion is unnecessary. Each codon is used with the grammar to decide which choice to make when replacing a non-terminal symbol using a given production rule. The first codon is used to select which of the production rules will replace the first non-terminal symbol. This is done by calculating the value of the codon *modulus* the number of production rules to choose from. As an example, for a codon with value 43 given 6 production rules, production rule 1 is chosen (note that the first production rule is at index 0) as 43 *mod* 6 is 1. If this production rule creates a sentence containing further non-terminal symbols the second codon is used to select the production rule for the first non-terminal set in the new sentence. This process is continued until the sentence is made up of only terminal symbols and the mapping process is complete. In this study, the sentences produced take the form of Java code rep-

resenting portions of low-level heuristics. A more detailed explanation of a GE system is provided by O'Neill and Ryan [14].

Recently, Burke et al. [22] have used Grammatical Evolution to generate low-level heuristics for bin packing. This paper generates heuristics which can consistently obtain solutions which use only one bin more than the optimal lower bound and often the optimal number of bins itself. GE was also seen to be suitably flexible enough to generate different move operators for different classes of bin packing problems as appropriate. Keller and Poli [23, 24] also use a grammar-based genetic programming system to evolve solvers for the travelling salesman problem.

4 Vehicle Routing Problems

The vehicle routing problem (VRP) is an NP-Complete [15] combinatorial optimisation problem where a number of customers are to be serviced by a fleet of vehicles subject to a number of constraints. Different objectives can be considered depending on the goal of the problem. Typical objectives include; minimisation of cost with respect to distance travelled, minimisation of the global travel time, minimisation of the number of vehicles required to service all customers, minimisation of the penalty costs associated with partial service of customers. The objective could also be a weighted combination of such objectives. Real-world commodity distribution in logistics is a complex problem with constrains varying depending on the application. It is therefore natural that many different variants of the VRP exist, each simplifying the problem to a smaller set of constraints which impose the most important restrictions in each specific application of the problem. A large number of exact [25, 26] and meta-heuristic [27, 28] methods have been applied in the literature to solve such problems.

Recently there has been an increasing gain of emphasis on solution methods which operate across different VRP variants. One of the state-of-the-art results obtained by such unified heuristics is the hyper-heuristic approach of Pisinger and Ropke [16]. This work is based on the Adaptive Large Neighbourhood Search (ALNS) framework initially presented by Ropke and Pisinger [29]. The proposed framework is a selection hyper-heuristic which when given a complete solution, traverses the search space through the application of heuristics which remove a number of requests from the solution and subsequently, heuristics to re-insert the removed requests. The selection of the next removal or insertion heuristic to use is based on statistical information gathered during the search. This work also provided a unified model for the VRP allowing five VRP variants to be tested following transformation to the Rich Pickup and Delivery Problem with Time Windows (RPDPTW). Here we will use this model to test our method on the two best known VRP variants; the vehicle routing problem with time windows (VRPTW) and the capacitated vehicle routing problem (CVRP).

5 Grammatical Evolution Hyper-heuristics for the VRP

Variable Neighbourhood Search (VNS) [30] is a well studied meta-heuristic methodology for global optimisation. A basic VNS algorithm is outlined in Algorithm 1.

Algorithm 1: Outline of a standard VNS algorithm

```
N: set of k neighbourhood structures, {N_1, N_2,...,N_k};
f: solution evaluation function;
x ← Construct initial solution;
repeat
    k ← 1;
    repeat
        Shaking: x' ← new point in neighbourhood N_k(x);
        Local Search: x'' ← result of local search from x';
        if f(x'') < f(x) then
            x ← x'' ;
            k ← 1 ;
        else
            k ← k + 1;
        end
    until k = kmax;
until stopping criteria met;
```

Operating on a complete solution initialised using a chosen method, VNS explores increasingly distant neighbours of the current solution using a predefined set of neighbourhood move operators. This process is known as 'shaking'. Following this, local search is performed to reach a local optimum from the point reached by the shaking procedure. The incumbent solution is replaced by a solution generated by a given neighbourhood move and subsequent local search if such a move will yields improvement in solution quality. This can be considered as a random descent, first improvement method. In the case where an improved solution is not found, the size of neighbourhood move is increased, thus effectively changing the neighbourhood structure used. This ensures the search is diversified sufficiently by performing increasingly larger neighbourhood moves in order to reach more promising areas of the search space when stuck in local optima.

Within this framework we will use grammatical evolution to generate the construction heuristic initialising a solution and ruin and insertion heuristics to perform the shaking procedure. Essentially we will evolve the order in which nodes are inserted into and removed from a solution through the use of a grammar. The grammar used is outlined in Figure 1. From the starting symbol <S>, three heuristic components are evolved using a single genome to select production rules. The set of terminal functions representing information fields which must be retrieved from the current solution state is shown in Table 1. Some of the information fields are not used by some problem variants and will con-

tain null values however it is still important to include such fields to enable the hyper-heuristic to generate heuristics across a broader class of routing problems. Those symbols prefixed 'rqi-' correspond to information about individual requests whilst those prefixed 'rti-' correspond to information information about a route.

```
                    <S> ::= <InitialSolution> <Ruin> <Recreate>
          <InitialSolution> ::= <Recreate> | (empty-solution)
                <Recreate> ::= <RecreateOrdered> | <RecreateStepwise>
          <RecreateOrdered> ::= <RequestFieldOp> <Order>
         <RecreateStepwise> ::= <Steps> <StepEnd>
                    <Ruin> ::= <RuinOrdered> | <RuinConditional> | <RequestSelection>
            <RuinOrdered> ::= <RouteSelectionLength> <RouteFieldOp> <Order> <RequestSelection>
         <RuinConditional> ::= <RouteFieldOp> <RelationalOp> <RouteFieldOp> <RequestSelection>
        <RequestSelection> ::= <RequestSelectionOrdered> | <RequestSelectionConditional>
<RequestSelectionOrdered> ::= <RequestFieldOp> <Order>
<RequestSelectionConditional> ::= <RequestFieldOp> <RelationalOp> <RequestFieldOp>
      <RouteSelectionLength> ::= (numroutes-RC) | (percentage-RC)
                      <op> ::= (add) | (sub) | (mul) | (div) | <MaxMin>
                   <Steps> ::= <NextStep> <Steps> | <NextStep>
                <NextStep> ::= <MaxMin> <RequestFieldOp>
                  <MaxMin> ::= (max) | (min)
                 <StepEnd> ::= (step-cycle) | (repeat-last)
            <RouteFieldOp> ::= <op> <RouteFieldOp> <RouteFieldOp> | <RouteField>
              <RouteField> ::= (rti-iuc) | (rti-d) | (rti-rc)
           <RequestFieldOp> ::= <op> <RequestFieldOp> <RequestFieldOp> | <RequestField>
             <RequestField> ::= (rqi-d) | (rqi-pat) | (rqi-pdt) | (rqi-puc) | (rqi-dat) | (rqi-ddt) |
                                (rqi-duc) | (rqi-prc) |(rqi-drc) | (rqi-pwt) | (rqi-pindx) |
                                (rqi-dwt) | (rqi-dindx) | (rqi-pst) | (rqi-ptws) | (rqi-ptwe) |
                                (rqi-pprevd) | (rqi-pnextd) | (rqi-dst) | (rqi-dtws) | (rqi-dtwe) |
                                (rqi-dprevd) | (rqi-dnextd)
                   <Order> ::= (ascending) | (descending)
            <RelationalOp> ::= (lt) | (gt) | (lte) | (gte) | (eq) | (neq)
```

Fig. 1. The grammar defining the components and structure of the heuristics

A standard set of non-terminal symbols is used to represent a number of basic binary arithmetic and relational operators shown in Table 2. Instead of the traditional divide function here we use protected divide. As there is always a possibility that the denominator could be zero, protected divide replaces zero with 0.001. In the case of relational operators the comparison is always made from left to right.

The constructive component of the heuristic constructs an initial feasible solution from an empty solution, it is also possible to leave the initial solution empty. The recreate component works in much the same way without the option of leaving the solution empty. Following the RPDPTW model unallocated requests are permitted however they are associated with a high penalty cost. Two methods for selecting the next request to insert are used, ordered and stepwise. *Ordered selection* uses a component composed of binary arithmetic operators and solution state information to rank each unallocated request. The order in which requests are inserted into the solution is derived from this ranking with the direction in which requests are considered determined by one of two terminal symbols, (ascending) and (descending). *Stepwise selection* evolves a sequence of different criteria to use at each 'step' when considering which request to insert. If the number of potential requests to insert is greater than the number of se-

Table 1. Set of terminal symbols which correspond to information request and route information within a solution

Symbol	Description
rqi-d	Commodity demand of a request
rqi-pat	Arrival time of the vehicle at the pickup node
rqi-pdt	Departure time of the vehicle at the pickup node
rqi-puc	Used vehicle capacity when leaving the pickup node
rqi-dat	Arrival time of the vehicle at the delivery node
rqi-ddt	Departure time of the vehicle at the delivery node
rqi-duc	Used vehicle capacity when leaving the delivery node
rqi-prc	Residual vehicle capacity when leaving the pickup node
rqi-drc	Residual vehicle capacity when leaving the delivery node
rqi-pwt	Time the vehicle must wait at the pickup node
rqi-pindx	The visit index of the pickup node within the route
rqi-dwt	Time the vehicle must wait at the delivery node
rqi-dindx	The visit index of the delivery node within the route
rqi-pst	Service time of the pickup node of the request
rqi-ptws	Opening time of the pickup node time window
rqi-ptwe	Closing time of the pickup node time window
rqi-pprevd	Distance between pickup node and previous node within the route
rqi-pnextd	Distance between the pickup and following node within the route
rqi-dst	Service time of the delivery node of the request
rqi-dtws	Opening time of the delivery node time window
rqi-dtwe	Closing time of the delivery node time window
rqi-dprevd	Distance between the delivery and previous node within the route
rqi-dnextd	Distance between the delivery and following node within the route
rti-iuc	The used capacity when vehicle leaves depot
rti-irc	The residual capacity when vehicle leaves depot
rti-d	Total distance of route

Table 2. Set of non-terminals which represent binary arithmetic and relational operators

Symbol	Description
add	Add two inputs
sub	Subtract second input from first input
mul	Multiply two inputs
div	Protected divide function
max	Maximum value between two inputs
min	Minimum value between two inputs
lt	Less than ($<$)
gt	Greater than ($>$)
lte	Less than or equal to ($\leq$)
gte	Greater than or equal ($\geq$)
eq	Equal
neq	Not equal

lection steps defined one of two options are available, (step-cycle) will return to the first step and cycle through the sequence of criteria again and (repeat-last) will re-use the last criteria in the sequence until all requests are inserted.

As the ruin phase works with a complete solution, selecting the routes from which to remove requests is not a trivial decision. A simple solution is to allow requests to be removed from any route however here we also allow the heuristic to evolve a subset of routes from which to choose. The number of routes to be selected is one of two random constants, either a number between 1 and the total number of routes (numroutes-RC) or a number between 0 and 1 representing the percentage of routes to be selected (percentage-RC). The order in which a subset of routes are considered is either ordered or conditional. *Conditional selection* iterates over the complete set of routes and returns a subset of routes which satisfy a condition set by a criteria evolved in the grammar. If number of routes selected is less than the number specified by the random constant, the remaining routes are selected randomly. The ruin heuristic is parametric with the number of requests to remove determined by the value of k taken from the overall VNS framework. Once the routes are selected the order in which requests are removed must be defined. Two methods for selecting the next request to remove are used, ordered selection as defined previously and conditional selection (as with the selection of routes however the iteration is performed over requests rather than routes). In the case of conditional selection, if less than k requests are selected using the evolved condition the remaining requests are removed randomly.

The parameters of the VNS search algorithm in which the generated construction, ruin and insertion heuristics operate, are set to initial $k = 1$, maximum $k = 30$ with k increased by 1 for each non-improving step. k is reset to 1 when an improving move is made. The local search used is a hill-climber which removes a request from an existing route, and relocates it to a different route so that the best improvement was achieved. Finally the stopping criterion used was 10 consecutive iterations of non-improvement after 30 non-improving steps. All experiments were performed in an offline manner i.e. a separate run of the GE system is performed on each individual instance. The parameters used in the GE runs are summarised in Table 3.

Table 3. Summary of Grammatical Evolution Parameters

Parameter	Value
Generations	50
Population Size	1024
Crossover Probability	0.9
Mutation Probability	0.05
Reproduction Probability	0.05
Maximum Tree Depth	17
Selection Method	Tournament Selection, size 7

6 Results

Table 4 shows the results of the GE hyper-heuristic (GE-PHH) on the first 10 instances for the CVRP from Augerat et al. [31] [1]. These instances contain either 5 or 6 vehicles and between 31 and 38 customers. The optimal solution is known for all of these instances and has been obtained by a number of methods in the literature [32]. 'Proximity' is calculated as $Optimal\ Value/Result\ Obtained$. To make some assessment of the generality of our method, experiments are also performed on instances taken from Solomon [33] with results shown in Table 5. There are three types of instance in this set; 'R' instances contain customers whose geographic locations have been randomly generated, 'C' instances comprise of clusters of customers and 'RC' instances consist of a mixture of both types of customers.

Table 4. Results of GE-PHH on the first 10 instances of Augerat et al. [31]

Instance Name	Optimal Value	GE-PHH (Vehicles)	Proximity
A-n32-k5	784	811.80 (5)	0.97
A-n33-k5	661	664.79 (5)	0.99
A-n33-k6	742	785.45 (6)	0.94
A-n34-k5	778	828.01 (5)	0.94
A-n36-k5	799	849.22 (5)	0.94
A-n37-k5	669	678.92 (5)	0.99
A-n37-k6	949	1020.06 (6)	0.93
A-n38-k5	730	826.83 (5)	0.88
A-n39-k5	822	905.23 (5)	0.91
A-n39-k6	831	838.75 (6)	0.99
Average			0.95

In all cases the best solutions obtained for each instance use the optimal number of vehicles. From these results we can see that the generated heuristics are able to reach promising regions of the search space however this does not necessarily lead to global optima. This limitation may be due to the nature of the local search operator used. As the local search only considers moving a request from one route to another. In some near optimal solutions requests must be exchanged within a single route to reach the global optimum. We observe that the application of the system on VRPTW does not seem to produce as high quality results as for the CVRP. This could be due to the set of components not being well suited to cover the temporal requirements of time window related problems. The system performs particularly poorly on the 'C102.100' instance having a negative impact on the average proximity.

[1] These instances and optimal solutions were taken from www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm

Table 5. Results of GE-PHH on a selection of instances from Solomon [33]

Instance Name	Optimal Value	GE-PHH (Vehicles)	Proximity
C101.100	827.3	902.64716 (10)	0.92
C102.100	827.3	1198.97254 (10)	0.69
R101.100	1637.7	1766.807 (20)	0.93
R102.100	1466.6	1596.96749 (18)	0.92
RC101.100	1619.8	1871.2241 (15)	0.87
RC102.100	1457.4	1771.4629 (14)	0.82
Average			0.86

7 Conclusions and Future Work

In this preliminary work we have shown that grammatical evolution shows potential as a hyper-heuristic to generate components of a VNS system to solve the VRP. This method is defined as a hyper-heuristic as it operates on a search space of heuristics rather than directly on a search space of solutions. To our knowledge, this is the first time in literature a GE hyper-heuristic has been used to solve the vehicle routing problem. This method has shown that automatically generating heuristics for the VRP could be an interesting future research direction. We are currently working on evolving each of the components in isolation rather than using a single genome and grammar to evolve the whole system. As mentioned in the previous section this method was restricted somewhat by the choice of local search operator. There are a large number of operators in the literature for the travelling salesman problem (TSP) which could be implemented to also allow request swaps within a route. There are also a variety of standard construction heuristics for the VRP in the literature. These could replace the constructive phase of our method leaving the focus on evolving the ruin and recreate heuristics within the VNS framework.

References

1. Ross, P.: Hyper-heuristics. In Burke, E.K., Kendall, G., eds.: Search Methodologies: Intrd. Tut. in Optimization and Decision Support Tec. Springer (2005) 529–556
2. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.: A Classification of Hyper-heuristics Approaches. In: Handbook of Metaheuristics 2nd ed. Springer (2010) 449–468
3. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. The MIT Press, Cambridge, MA (1992)
4. Burke, E.K., Hyde, M., Kendall, G., Woodward, J.: A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. IEEE Transactions on Evolutionary Computation **14**(6) (2010) 942–958
5. Burke, E.K., Hyde, M.R., Kendall, G.: Evolving bin packing heuristics with genetic programming. In: PPSN 2006. Volume 4193 of LNCS., Springer (2006) 860–869

6. Burke, E.K., Woodward, J., Hyde, M., Kendall, G.: Automatic heuristic generation with genetic programming: Evolving a jack-of-alltrades or a master of one. In: GECCO 2007. (2007) 1559–1565
7. Burke, E.K., Hyde, M., Kendall, G., Woodward, J.: Automating the packing heuristic design process with genetic programming. Evolutionary Computation **20**(1) (2012) 63–89
8. Geiger, C.D., Uzsoy, R., Aytug, H.: Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. Journal of Scheduling **9**(1) (2006) 7–34
9. Kumar, R., Joshi, A.H., Banka, K.K., Rockett, P.I.: Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In: GECCO 2008, ACM (2008) 1227–1234
10. Bader-El-Den, M., Poli, R.: Generating sat local-search heuristics using a gp hyperheuristic framework. In Monmarch, N., Talbi, E.G., Collet, P., Schoenauer, M., Lutton, E., eds.: Artificial Evolution. Volume 4926 of LNCS., Springer Berlin / Heidelberg (2008) 37–49
11. Fukunaga, A.S.: Automated discovery of composite sat variable-selection heuristics. In: Artificial intelligence. (2002) 641–648
12. Fukunaga, A.S.: Evolving local search heuristics for sat using genetic programming. In: GECCO 2004. Volume 3103 of LNCS., Springer-Verlag (2004) 483494
13. Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation **16**(1) (2008) 31–61
14. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language. Volume 4 of Genetic programming. Kluwer Academic Publishers (2003)
15. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1979)
16. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Computers and Operations Research **34**(8) (2007) 2403–2435
17. Fisher, M., Thompson, G.: Probabilistic learning combinations of local job-shop scheduling rules. In: Factory Scheduling Conference. (1961)
18. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: PATAT 2000, London, UK, Springer-Verlag (2001) 176–190
19. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyperheuristics: A survey of the state of the art. Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham (2010)
20. John H. Drake, Matthew Hyde, K.I., Özcan, E.: A genetic programming hyperheuristic for the multidimensional knapsack problem. In: CIS 2012. (2012) 76–80
21. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: Exploring Hyper-heuristic Methodologies with Genetic Programming. In: Computational Intelligence: Collaboration, Fusion and Emergence. Springer-Verlag (2009) 177–201
22. Burke, E.K., Hyde, M.R., Kendall, G.: Grammatical evolution of local search heuristics. IEEE Transactions on Evolutionary Computation **16**(3) (2012) 406–417
23. Keller, R.E., Poli, R.: Linear genetic programming of metaheuristics. In: GECCO 2007, ACM (2007) 1753–1753
24. Keller, R.E., Poli, R.: Linear genetic programming of parsimonious metaheuristics. In: CEC 2007. (2007) 4508–4515

25. Laporte, G.: The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research **59**(3) (1992) 345 – 358
26. Toth, P., Vigo, D.: Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics **123**(13) (2002) 487 – 512
27. J-F Cordeau, M. Gendreau, G.L.J.Y.P., Semet, F.: A guide to vehicle routing heuristics. The Journal of the Operational Research Society **53**(5) (2002) 512–522
28. Bräysy, O., Gendreau, M.: Vehicle routing problem with time windows, part ii: Metaheuristics. Transportation Science **39**(1) (2005) 119–139
29. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science **40**(4) (2006) 455–472
30. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research **24**(1) (1997) 1097–1100
31. Augerat, P., Rinaldi, G., Belenguer, J., Benavent, E., Corberan, A., Naddef, D.: Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical report, RR 949-M, Universite Joseph Fourier, Grenoble (1995)
32. Ralphs, T., Kopman, L., Pulleyblank, W., Jr, L.T.: On the capacitated vehicle routing problem. Mathematical Programming Series B **94** (2003) 343359
33. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research **35**(2) (1987) 254–265