Evolution of Hyperheuristics for the Biobjective 0/1 Knapsack Problem by Multiobjective Genetic Programming

Rajeev Kumar*
Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Kharagpur, WB 721302, India
rkumar@cse.iitkgp.ernet.in

Ashwin H. Joshi
Dept. of Computer Sc. & Engineering
Indian Institute of Technology Kharagpur
Kharagpur, WB 721302, India
ajoshi@cse.iitkgp.ernet.in

Krishna K. Banka
Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Kharagpur, WB 721302, India
kkbanka@cse.iitkgp.ernet.in

Peter I. Rockett
Laboratory for Image & Vision Engineering
Dept. of Electronic & Electrical Engineering
University of Sheffield
Mappin Street, Sheffield S1 3JD, UK
p.rockett@shef.ac.uk

ABSTRACT

The 0/1 knapsack problem is one of the most exhaustively studied NP-hard combinatorial optimization problems. Many different approaches have been taken to obtain an approximate solution to the problem in polynomial time. Here we consider the biobjective 0/1 knapsack problem. The contribution of this paper is to show that a genetic programming system can evolve a set of heuristics that can give solutions on the Pareto front for multiobjective combinatorial problems. The genetic programming (GP) system outlined here evolves a heuristic which decides whether or not to add an item to the knapsack in such a way that the final solution is one of the Pareto optimal solutions. Moreover, the Pareto front obtained from the GP system is comparable to the front obtained from other human-designed heuristics. We discuss the issue of the diversity of the obtained Pareto front and the application of strongly-typed GP as a means of obtaining better diversity.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Stochastic programming; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program Synthesis*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*Heuristic Methods*

General Terms

Algorithm, Design, Experimentation.

*The author gratefully acknowledges receipt of conference travel support from Google Inc.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA

Keywords

Optimization methods, multiobjective optimization, genetic algorithm, genetic programming, heuristics, combinatorial optimization, 0-1 knapsack problem, Pareto front.

1. INTRODUCTION

The single objective 0-1 knapsack is a well-known combinatorial optimization problem in which, given a set of n items with a profit, P and a weight, W associated with each item, and a knapsack of capacity, C, the goal is to select the items such that total profit is maximized with the constraint that the total weight of the selected items should not exceed the knapsack capacity, C.

In this paper, the biobjective variant of this problem is studied in which the two objectives are to:

1. Maximize the total profit
2. Minimize the total weight of the selected items

without any additional constraint. The knapsack problem frequently arises in resource allocation within budget constraints and is therefore of great practical importance. In real world applications, the decision maker should have all the possible options to hand which is best achieved by considering the problem in the multiobjective domain, i.e. by generating the set of Pareto-optimal solutions.

Evolutionary approaches have previously been applied to the family of knapsack problems – see for example, [22] – but these directly evolve a set of solutions using a pre-specified and human-designed heuristic. In contrast, here we focus on evolving a set of heuristics which will *directly* produce the Pareto set of solutions to the knapsack problem; crucially, we do not evolve *solutions* to the knapsack problem. We present a *hyperheuristic* which evolves 'rules' to solve a multiobjective combinatorial optimization problem. We show how a heuristic naturally evolves using genetic programming without any human intervention. The quality of evolved heuristic is comparable to a range of the human-designed heuristics and can be reused on new problem instances without any

further computation-intensive evolution, unlike other evolutionary approaches. A major advantage of this approach is that we need not solve every new instance of the problem – the evolved hyperheuristics can be straightforwardly reused to solve new problem instances. We also attempt to show the impact of strongly-typed genetic programming on the evolution and the quality of heuristics that are evolved.

In Section 2 we discuss previous research into solving the knapsack problems and present a formal definition in Section 3. We describe the implementation of the genetic programming framework in Section 4 together with results. In Section 5 we discuss various strategies for improving the diversity of the generated front of Pareto-optimal solutions. The paper concludes with Section 6 in which we discuss future research directions.

2. RELATED WORK

2.1 Existing Approaches

The multiobjective 0-1 knapsack problem is a well-studied problem and many variants of it are available in the literature. Even the single objective case has been proven to be NP-hard. In general, the multiobjective variants of the problem are even harder than the single objective case. Due to their practical importance, the family of knapsack problems has been the subject of a great deal of work in the past.

Ibarra & Kim [14] have given the fully polynomial time approximation (FPTAS) scheme for the single objective problem. Lawler [23] has reported a fully polynomial fast approximation scheme while Magazine & Oguz [24] have presented a fully polynomial approximation algorithm for the single objective case.

For a single objective m-dimensional knapsack problem, a polynomial time approximation scheme (PTAS) was presented by Frieze & Clarke [10]. Erlebach et al. [8] described a practical FPTAS for the multiple one-dimensional knapsack problem and for the m-dimensional knapsack problem; they also described a polynomial time approximation scheme based on linear programming. Hanafi [12] has studied the bounds and computational aspects of the multidimensional 0-1 knapsack problem (MKP); Hanafi's paper also includes a survey of the recent literature on the theoretical aspects as well as exact or approximate solutions. Chekuri & Khanna [4] have given a PTAS for the MKP and have compared the performance of their scheme with that of the generalized assignment problem. This same problem has also been approached by Hembecker et al. [13] using particle swarm optimization. Chu & Beasley [5] have presented a scheme based upon genetic algorithms which utilizes a heuristic operator embedding domain-specific knowledge into a standard genetic algorithm approach. Much research has been performed over the decades and the problem continues to be a challenging area of research.

Zitzler & Thiele [27] pioneered solving multiobjective 0-1 knapsack problems using EAs. They formulated the problem using m knapsacks and maximized the profits simultaneously for all m knapsacks within weight constraints. Later, many other researchers (e.g. [25], [15], [16]) attempted to solve the same problem formulation using other variants of EAs. Gandibleux & Freville [11] have also addressed the multiobjective variant of the m-dimensional knapsack problem using tabu search. Barichard & Hao [1] have studied another multiobjective variant of the knapsack problem in which multiple objectives were considered along with more than one constraint. They have followed a hybrid approach and introduced GTS (genetic tabu search) which combined a genetic procedure and a tabu search operator.

The multiobjective variant that we have considered here has been discussed in [22]. In the literature, this problem has been solved using several approaches, such as deterministic heuristics [26], evolutionary algorithms (EA), etc. Recently this biobjective variant of the problem has been discussed by Kumar & Singh [22] and thoroughly analyzed in [19, 20].

Little work exists in the literature on the *evolution of heuristics* for combinatorial multiobjective optimization problems using genetic programming. Most notably, Burke et al. [2, 3] have recently described an approach to evolve a heuristic for the single objective bin packing problem using genetic programming. They have considered an online bin packing problem and obtained a heuristic similar to and of comparable performance to the well-known first-fit heuristic.

The present paper investigates the use of multiobjective genetic programming (MOGP) to produce hyperheuristics for multiobjective combinatorial optimization problems in general, and the biobjective knapsack problem in particular.

3. PROBLEM FORMULATION

The knapsack problem is described by a knapsack of size C and n items with three sets of variables related to the items:

1. Decision variables x_1, x_2, ...,x_n, where $x_i \in [0, 1]$ and x_i describes either the omission or inclusion of the i-th item in the knapsack.

2. Weights W_1, W_2, ..., W_n, where $W_i > 0$ and W_i is the weight of the i-th item.

3. Profits P_1, P_2, ..., P_n, where P_i denotes the profit from including the i-th item in the knapsack.

In addition, $W_i, P_i \in \mathbb{N}$. The *single-objective* knapsack problem can be formally stated as:

$$\text{Maximize} \sum_{i=1}^{n} P_i x_i$$

subject to the constraint that $\sum_{i=1}^{n} W_i x_i \leq C$.

The *biobjective* formulation of the problem can be defined as:

$$\text{Maximize} \sum_{j=1}^{n} P_j x_j \quad \text{and} \quad \text{Minimize} \sum_{j=1}^{n} W_j x_j$$

That is, we aim to maximize the total profit while simultaneously minimizing the total weight of the knapsack. This is clearly a multiobjective problem since a number of possible solutions exist which trade-off the non-commensurable objectives of profit and weight.

Since our goal has been to evolve heuristics which are *generally useful*, we gauge the fitness of a potential heuristic over a set of q representative knapsack problems. In terms of their practical implementation, the values of the biobjective fitness function are given by:

- Profit: 1 - $\frac{1}{q}\sum_{i=1}^{q}\left[\frac{\sum_{j=1}^{n}P_{i.j}x_{i,j}}{\sum_{k=1}^{n}P_{i,k}}\right]$
- Weight: $\frac{1}{q}\sum_{i=1}^{q}\left[\frac{\sum_{j=1}^{n}W_{i,j}x_{i,j}}{\sum_{k=1}^{n}W_{i,k}}\right]$

where n is the number of selected items in i-th test problem.

Here $P_{i,j}$ and $W_{i,j}$ represent the current profit and weight, respectively and $x_{i,j}$ is a decision variable which indicates whether the object is included in the knapsack or not. The normalizations (inside the square brackets) by total profit = $\sum_{k=1}^{n}P_{i,k}$ and the total weight = $\sum_{k=1}^{n}W_{i,k}$, give measures of what fractions of the available profit/weight the heuristic is able to exploit.

Finally, the fitness values are normalized between 0 and 1, with 0 indicating the best and 1 indicating the worst value. With this formulation of the objectives, the optimization becomes a minimization in both objectives.

4. MOGP IMPLEMENTATION

The individuals in the population used in genetic programming are trees consisting of functions and terminals suitable for the problem domain. Some of these operators and terminals are universal in the sense that they are needed in most problem domains, whereas others are specific to the problem domain. Arithmetic functions such as addition and multiplication are examples of the first; an operator that, say, selects the minimum weight from a set of weights is an example of the latter.

The MOGP framework used here is a straightforward adaptation to genetic programming of the Pareto Converging Genetic Algorithm (PCGA) of Kumar & Rockett [21]. This is a (μ+2) steady-state evolutionary algorithm in which we always maintain the best solutions obtained so far and look to replace the two weakest solutions in the population with newly-obtained solutions. We have used the same multiobjective ranking/selection scheme as Fonseca & Fleming [9].

The output of an individual tree in the population evaluated on a given knapsack item determines whether or not that item should be added to the knapsack; the tree inputs are the profit and the weight of the item. By evolution, the system learns optimal heuristics from the training data. It should be noted that there is no manual intervention and no constraints put on the evolution of the heuristic. We input only the necessary parameters such as current profit and weight and the heuristic emerges in a natural way.

We have used the usual arithmetic operators of: $+$, $-$, $\times$, $\div$ and the comparison operators, $\leq$ and $\geq$. For the division operator, we have used 'protected' division [18].

The initial population was created using the ramped half-and-half method in which half the population was recursively grown at random until either a terminal is selected to complete a subtree, or the maximum tree depth is reached. The other half of the population is produced in a similar manner except a terminal is debarred from being selected until the maximum tree depth has been attained. The initial population is therefore diverse and comprises trees of a variety of 'shapes'. Here we have used an initial maximum tree depth of 5.

Bloat – the excessive growth in tree size – is known to be a significant issue in GP. Here we have imposed a maximum value on the tree depth, although more elaborate schemes are possible. These remain an area for future study.

We carried-out the experiments on different datasets of 100, 250, 500 and 750 knapsack items. The datasets were produced in the same way as Zitzler & Thiele [27] where values of profits and weights were generated randomly for each dataset. Similarly, the testing was performed on randomly generated datasets. We performed experiments on different ranges of profits and weights up to 100.

Evolution was continued for a fixed number of tree evaluations.

4.1 Overall Program Structure

The pseudocode for the overall program structure is shown in Algorithm 1, where *Evaluate* denotes the (biobjective) fitness evaluation of an individual GP tree in the population. The pseudocode to compute the fitness function is shown in Algorithm 2.

Algorithm 1 Basic Program Structure

```
for each evolved program a in archive A do
  for each object i ∈ input I do
    output = Evaluate(a, P_i, W_i)
    if output ≥ 1.0 then
      pick_object(i)
    end if
  end for Output Total profit and Total Weight as Pareto
  solution
end for
```

Here *tree* refers to the current individual in the population. This pseudocode refers to the fitness evaluation of a single individual. The fitness of all individuals in the population are calculated similarly.

Expressed in words, the fitness of an individual is evaluated by running over each item in the set of knapsack problems. If the output of the tree for a given item is greater than a threshold, then that item is added to the knapsack and the profit and weight of the current configuration adjusted accordingly. After considering all items, the profit and weight objectives are normalized, as set-out in Section 3.

Algorithm 2 Fitness Function Evaluation

```
profitFitness = 0
weightFitness = 0
for each knapsack instance t ∈ Training Set T do
  totalProfit = 0
  totalWeight = 0
  for each knapsack item i ϵ t do
    output = Evaluate(P_i, W_i)
    if output ≥ 1 then
      totalProfit = totalProfit + P_i
      totalWeight = totalWeight + W_i
    end if
  end for
  profitFitness = profitFitness+
  totalProfit/MaxTotalProfit
  weightFitness = weightFitness+
  totalWeight/MaxTotalWeight
end for
profitFitness = 1 − profitFitness/NoOfInstances
weightFitness = weightFitness/NoOfInstances
```

4.2 Conventional Comparators

Clearly our longer-term motivation is demonstrate the effective evolution of heuristics to solve the general class of NP-hard multiobjective combinatorial optimization problems; our current use of the biobjective knapsack problem is a convenient and much-studied demonstrator. To gauge the performance of the heuristics evolved here, we use the highly effective P/W heuristic [22], and dynamic programming [26] as benchmark comparators.

In the P/W algorithm, the items are arranged in descending order of their profit-to-weight ratio. Items are selected for inclusion in the knapsack in this order. The total complexity of the P/W algorithm is $\mathcal{O}(n \log\ n)$ and it yields n *optimal* points for an n-item knapsack. The dynamic programming algorithm [26], however, yields many more optimal points. Since the outputs from these two comparator algorithms are identical, we only make reference to the P/W results in the following.

4.3 Results and Discussion

We carried-out the experiments on different data sets of 100, 250, 500 and 750 possible knapsack items. The parameters used for the genetic programming are as shown in Tables 1 and 2.

Table 1: Common GP Parameters

Functions used	$+, -, \times, \div, \leq, \geq$
Terminals	P_i, W_i,and constants
Probability of Crossover	0.9
Probability of Mutation	0.1
Tree Initialization Method	Ramped half-and-half

Table 2: GP Parameters

#Items	100	250	500	750
Population Size	500	500	1000	2000
Maximum Tree Depth	5	7	7	8
Maximum Tree Evaluations	1000	1000	1500	2000

Figure 1 shows the comparison between the two approximated Pareto fronts produced by the human-derived P/W heuristic and the hyperheuristics evolved here using MOGP. It is clear that the solutions obtained from the evolved heuristics lie on, or very close to, the same curve as the P/W heuristic, in addition to giving a reasonable degree of sampling of the curve. It is evident, however, that there are certain regions of the curve in which no solutions have been generated. This is a consequence of premature convergence.

Three examples of the evolved trees are shown in Figure 2 from which it can be seen that the discovered heuristics are fairly simple and mostly well within the limit on maximum tree depth imposed during evolution.

4.4 Quantitative Analysis

We have quantitatively evaluated and assessed the convergence, diversity and extent of the Pareto front obtained using various evaluation metrics such as: the C-measure [27], convergence [7], spread [6] and hypervolume [17]. The values obtained are shown in Table 3. In the case of hypervolume, we have taken the ratio of the hypervolume-values obtained for the GP-front and the heuristic front. All values were computed by averaging over the test data.

Table 3: C-measure, Convergence, Spread and Hypervolume (S-measure) values for knapsack instance of 100 items. The results are compared against P/W heuristic values

#Items	100	250	500	750
C-measure	0.2138	0.4198	0.3698	0.4062
Convergence	0.0056	0.0023	0.0013	0.0011
Spread	1.2656	1.2242	1.3305	1.3254
Hypervolume	0.9944	0.9918	0.9936	0.9921

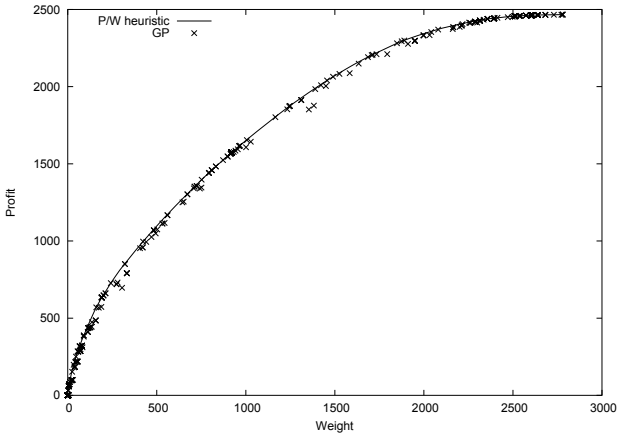

Figure 1: Plots for 100 items with max.depth of 5.

5. IMPROVING DIVERSITY

As can be seen from Figure 1, even though there is good convergence and quite good diversity, there are certain regions on the front that remain unfilled. That means we are obtaining rapid convergence but only to specific areas on the Pareto front, thus leaving gaps on the front. To improve the results and obtain a good, diverse sampling of the Pareto front, several experiments were performed and different measures were applied to improve diversity.

5.1 Enforcing Diversity

Initially the maximum range of values which profit and weight of an object can take was lowered to 20 in the hope that more diverse solutions could be obtained. For a maximum depth of 5 this did not show any significant improvement but increasing the maximum depth to 7 gave the improved results shown in Figure 3. This, however, is not a generally applicable solution to the problem of improving diversity since in many situations, the range of profits and weights is dictated by factors outside our control We thus explored three other measures aimed at improving the filling of the front.

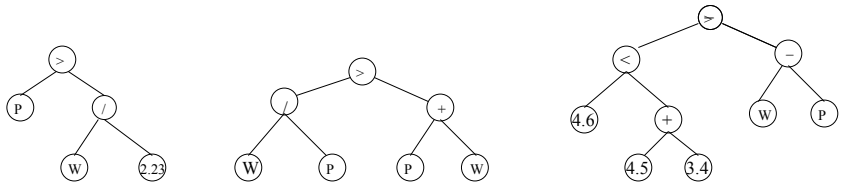

Figure 2: Example trees.

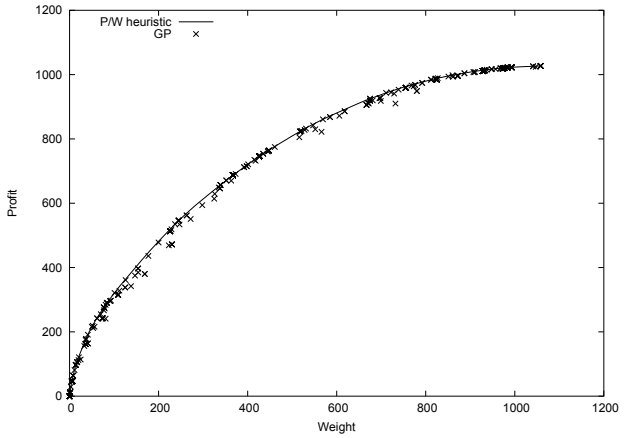

Figure 3: Results obtained with reduced range of profit and weight. (100 items)

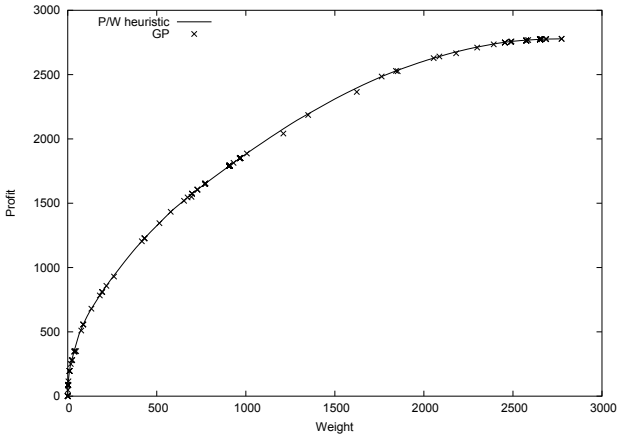

Figure 4: Results obtained with solutions forced in the middle region. (100 items)

5.1.1 *Forced Diversity*

It can be seen from the front in Figure 1, that there are many solutions concentrated towards the two ends of the curve but fewer solutions lying in the central region. To overcome this problem a *forced* diversity measure was tried in which after half the maximum number of tree evaluations (by which time we expected some solutions would have converged near the ends) we imposed a constraint on the generated solutions that they be in the middle region of the curve. That means we would only accept a new solution if it lay in the central zone. Otherwise we would reject it and generate another new solution and repeat, until we found one which was acceptable. We expected that the empty regions on the front would be filled by this forced diversity measure. Unfortunately, we observed that although it increased the number of solutions in the central zone, most of them converged to the same points which we had already obtained in the earlier experiments in Section 4.3 . There was thus no significant improvement, as is shown in the Figure 4 – in fact the diversity was degraded. Quite why the diversity was reduced is an area for future research.

5.1.2 *Reduced P/W Ratio*

Our second attempt at improving coverage of the front was to classify the input on the basis of profit-to-weight (P/W) ratio. We conjectured that if the inputs for both training (and testing) were classified on the basis of their profit-to-weight (P/W) ratio in a definite range then the results would improve. In fact, we observed that the solutions became *less* diverse. This happened because as we restrict the inputs to smaller ranges, then the P/W ratio-values for different items become very close and in many cases, equal. It thus becomes very difficult (or indeed impossible) to produce trees to correctly allocate the items. The degraded diversity is apparent in Figure 5.

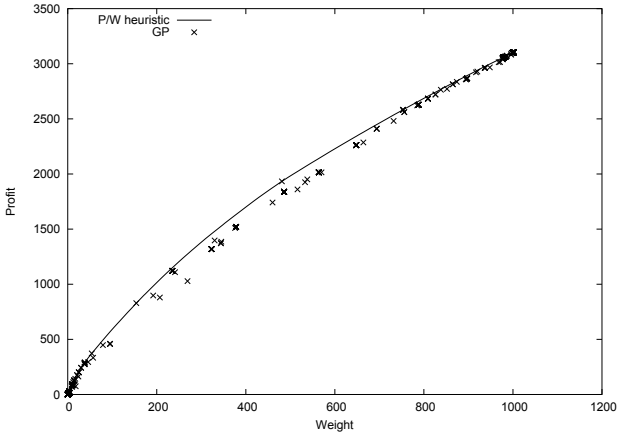

Figure 5: Results obtained with a reduced P/W range. (100 items)

5.1.3 *Crowding*

Finally, we tried to facilitate a diversity maintenance mechanism in the algorithm itself. In the basic PCGP algorithm the newly generated offspring can replace the two

weakest solutions in the population (providing they dominate them). There is no checking for diversity – although this generates good and quick convergence, it does not guarantee diversity. To overcome this a diversity check was used from the very beginning of the run: Whenever new offspring were generated, instead of selecting the two weakest solutions for possible replacement, the two solutions which lay in the most crowded, most densely-populated region were selected to be candidates for replacement. This means that we might replace two Pareto-optimal solutions with weaker, dominated solutions but this should only delay convergence. Unfortunately, this too did not give any significant improvement in the filling of the Pareto front.

We conclude that incorporating an effective phenotypic diversity preserving mechanism is difficult, at least in this problem.

5.2 Strongly-Typed Program Evolution

In order to achieve a good diversity, we followed another approach of evolving the trees by assigning *types* to them. Since the trees are typed rather than random, this helps in evolving more meaningful trees.

Just like in a strongly-typed programming language where each operand is associated with a specific type and each operator is constrained to take operands only of a given type or its subtypes, in strongly-typed genetic programming each terminal is associated with a given type and each function is constrained to take terminals of a certain type or its subtypes, and to return the result of a certain type. These typing constraints ensure that strongly-typed programs are evolved and are applied at every stage in genetic programming where they are necessary: Random creation of the initial population, crossover and mutation.

In the case of random creation of the initial population, in which an individual is created recursively by first randomly selecting the root and then randomly creating its subtrees, the typing constraint translates into selection of the function at the root of the desired type and the creation of subtrees only of the type that can be handled by that function. Without the use of typing constraints in the creation of the initial population, a large fraction of the population will comprise trees in which functions have illegal operands. That fraction will be larger for larger trees.

In the case of typed crossover, the typing constraint is enforced by exchanging randomly selected subtrees of the parents *only* if their types match. If this is not the case, selection of the splicing point is repeated until subtrees of compatible types are found. In the case of mutation, the selected subtree is replaced by a randomly-created tree of the matching type.

We have experimented with the evolution of trees by considering both typed as well as non-typed trees. Strongly-typed genetic programming (STGP) introduces typed functions into the GP genome. Type checking also reduces the search space, which is likely to improve the search. It makes little sense to recombine the two different types of data with crossover. Also, the initial population is much more meaningful when we use type checking and that helps in achieving good diversity because the solutions are obtained over a wide range. The implementation details for the typed trees are as shown in Table 4.

With the use of typing constraints, a significant improvement in the coverage of the results was obtained, as evidenced by the improved results shown in Figures 6, 7 and 8, for 100, 250 and 750 item problems, respectively. Also, since crossover now yields only type-correct trees there is a greater number of meaningful solutions. Thus solutions on the entire Pareto front are obtained without any significant gaps. The probability of successful crossover is, however, reduced as the types of subtrees selected must match, necessitating a larger (average) number of attempts before a successful crossover is achieved. Nonetheless, as evidenced by Figures 6, 7 and 8, strong typing is highly effective at ensuring diversity.

Table 4: Functions and Terminals considered for typed implementation

Type	Operators	Terminals
Numeric	$+$, $-$, $\times$, $\div$	P, W and constants
Logical	$\leq$, $\geq$	True, False

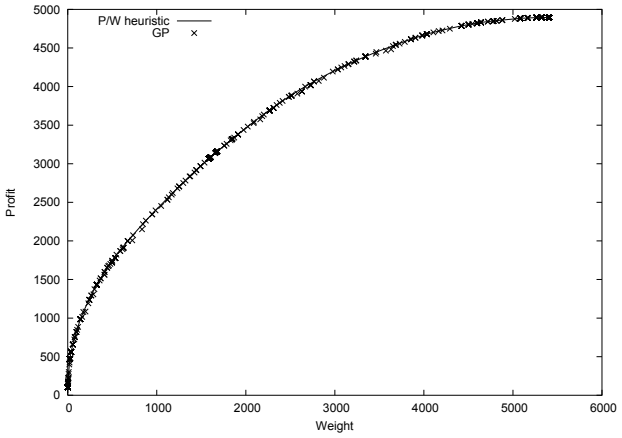

Figure 6: Improved results for strongly-typed trees. (100 items).

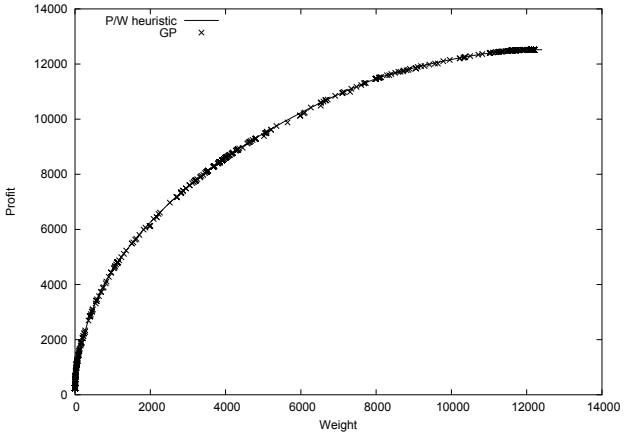

Figure 7: Improved results for strongly-typed trees. (250 items).

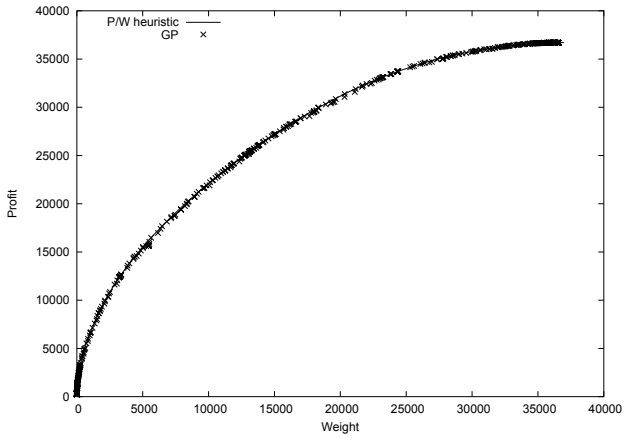

Figure 8: Improved results for strongly-typed trees. (750 items).

A quantitative analysis of the strongly-typed results is shown in Table 5; the results corresponding to the basic (untyped) algorithm of Section 4 have already been shown in Table 3.

Table 5: C-measure, Convergence, Spread and Hypervolume (S-measure) values for the strongly typed approach (100 items) compared against P/W heuristic values

#Items	100	250	500	750
C-measure	0.0300	0.0320	0.0339	0.0500
Convergence	0.0002	0.0001	0.0001	0.0001
Spread	1.9179	1.8587	1.8300	1.8162
Hypervolume	1.0000	0.9965	0.9961	0.9953

6. CONCLUSIONS AND FUTURE WORK

In this work we have shown that a set of heuristics can be evolved for the biobjective knapsack problem that gives a set of Pareto-optimal solutions. The Pareto fronts obtained in this way are indistinguishable from the Pareto fronts obtained using a human-designed heuristics – namely, the profit-to-weight ratio (P/W) heuristic. Moreover, these heuristics are evolved automatically without problem-specific knowledge and without human intervention.

As to future extensions of this work, we can anticipate evolving heuristics for other multiobjective combinatorial optimization problems such as multiobjective versions of the minimum spanning tree and traveling salesperson problems. This should lead to the identification of domain-specific basic building blocks for the GP system that will be useful in other related problems.

7. ACKNOWLEDGMENTS

We are grateful to Dr Yang Zhang for generously supplying the genetic programming code used in this study.

8. REFERENCES

[1] V. Barichard and J.-K. Hao. Genetic tabu search for the multi-objective knapsack problem. *Journal of Tsinghua Science and Technology*, 8(1):8–13, 2003.

[2] E. K. Burke, M. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *Proceedings of the 9^{th} International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, LNCS 4193, pages 860–869, Reykjavik , Iceland, September 2006. Springer Berlin / Heidelberg.

[3] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In *Genetic and Evolutionary Computation Conference (GECCO-2007)*, pages 1559–1565, London, UK, July 2007.

[4] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *Proc. of 11th Annual ACM-SIAM symposium on Discrete Algorithms*, pages 213–222, 2000.

[5] P. Chu and J. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, June 1998.

[6] K. Deb. *Multiobjective Optimization Using Evolutionary Algorithms*. Chichester, UK: Wiley, 2001.

[7] K. Deb and S. Jain. Running performance metrics for evolutionary multi-objective optimization. In L. Wang et al., editors, *Proceedings of the 4^{th} Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, volume 1, pages 13–20, November 2002, Nanyang Technical University, Singapore.

[8] T. Erlebach, H. Kellerer, and U. Pferschy. Approximating multiobjective knapsack problems. *Management Science*, 48:1603–1612, 2002.

[9] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, *5^{th} International Conference of Genetic Algorithms*, pages 416–423. Morgan Kaufmann, 1993.

[10] A. Frieze and M. Clarke. Approximation algorithms for m-dimensional 0-1 knapsack problem: Worst case and probabilistic analysis. *European Journal, Operations Research*, 15:100–109, 1984.

[11] X. Gandibleux and A. Freville. Tabu search based procedure for solving the 0-1 multi-objective knapsack problem: The two objectives case. *Journal of Heuristics*, 6(3):361–383, August 2000.

[12] F. S. Hanafi. The multidimensional 0-1 knapsack problem-bounds and computational aspects. *Annals of Operations Research*, 139(1):195–227, October 2005.

[13] F. Hembecker, H. S. Lopes, and W. G. Jr. Particle swarm optimization for the multidimensional knapsack problem. *Lecture Notes in Computer Science*, 4431:358–365, 2007.

[14] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problem. *Journal of ACM*, 22:463–468, 1984.

[15] A. Jaszkiewicz. On the performance of multiobjective genetic local search on the 0-1 knapsack problem - A comparative experiment. *IEEE Transactions on Evolutionary Computation*, 6:402–412, 2002.

[16] J. Knowles and D. Corne. M-PAES: A memetic algorithm for multiobjective optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 325–332, California, USA, 6-9 2000. IEEE Press.

[17] J. Knowles and D. Corne. On metrics for comparing nondominated sets. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 711–716, Piscataway, New Jersey, May 2002. IEEE Press.

[18] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[19] R. Kumar and N. Banerjee. Running time analysis of a multiobjective evolutionary algorithm on simple and hard problems. In *Proceedings of Foundations of Genetic Algorithms(FOGA-05)*, volume 3469 of *LNCS*, pages 112–131. Springer, 2005.

[20] R. Kumar and N. Banerjee. Analysis of a multiobjective evolutionary algorithm on the 0-1 knapsack problem. *Theoretical Computer Science*, 358(1):104–120, July 2006.

[21] R. Kumar and P. I. Rockett. Improved sampling of the Pareto-front in multiobjective genetic optimization by steady-state evolution: A Pareto converging genetic algorithm. *Evolutionary Computation*, 10(3):283–314, July 2002.

[22] R. Kumar, P. K. Singh, A. P. Singhal, and A. Bhartia. Evolutionary and heuristic algorithms for 0-1 knapsack problem. In A. Tiwari et al., editors, *Applications of Soft Computing*, volume 36 of *Advances in Soft Computing*, pages 331–340. Springer, 2006.

[23] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, November 1979.

[24] M. J. Magazine and O. Oguz. A fully polynomial approximation algorithm for the 0-1 knapsack problem. *European Journal of Operational Research*, 8(3):270–273, November 1981.

[25] G. Raidl. An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proc. IEEE Conference on Evolutionary Computation.*, pages 207–211, 1998.

[26] S.Goddard. Dynamic programming for 0-1 knapsack problem. `http://www.cse.unl.edu/~goddard/Courses/CSCE310J/Lectures/Lecture8-DynamicProgramming.pdf`.

[27] E. Zitzler and L.Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Trans. Evolutionary Computation*, 3:257–271, 1999.