Cooperative Strategies for Solving the Bicriteria Sparse Multiple Knapsack Problem

F. Sibel Salman
Graduate School of Industrial Administration
Garnegie Mellon University
Pittsburgh, PA 15213
fs2c@andrew.cmu.edu

Jayant Kalagnanam and **Sesh Murthy**
T.J. Watson Research Center
International Business Machines
Yorktown Hts, NY 10598
jayant/murthy@watson.ibm.com

Abstract-

For hard optimization problems, it is difficult to design heuristic algorithms which exhibit uniformly superior performance for all problem instances. As a result it becomes necessary to tailor the algorithms based on the problem instance. In this paper, we introduce the use of a cooperative problem solving team of heuristics that evolves algorithms for a given problem instance. The efficacy of this method is examined by solving six difficult instances of a *bicriteria sparse multiple knapsack problem***. Results indicate that such tailored algorithms uniformly improve solutions as compared to using predesigned heuristic algorithms.**

1 Introduction

The problem of assigning a given set of orders to the production units in the inventory arises frequently in production planning and scheduling. The objectives are both maximizing the total amount of orders that are assigned, and minimizing total waste due to the unused portion of the production units. Manufacturability considerations such as the compatibility of orders and production units in terms of quality, size, etc. impose additional assignment constraints. As production operations involve more complex processes and a larger product variety, the problem becomes more constrained. The *bicriteria sparse multiple knapsack problem* that we consider in this study is motivated by this application.

In this paper we focus on the use of a team of heuristic algorithms which cooperate to generate non-dominated solutions for this problem in a short computation time. Although there exist several heuristic approaches for solving multiple knapsack problems there does not exist a single dominant algorithm. Moreover, the performance of the heuristics vary by problem instance and as a result a specific heuristic will often demonstrate poor aggregate performance over a set of problem instances. However, if the heuristics were allowed to cooperate with each other so that

- the solution generated by one heuristic can be subsequently improved by another or
- the most appropriate subset of heuristics can be used to construct solutions for a given problem instance

then the aggregate performance of a collection of cooperating heuristics over a set of problem instances may be greatly improved. For this purpose we have developed a collection of fast heuristics and incorporated them in an A-team architecture, which provides a computational framework for implementing cooperation strategies among heuristics. We present results of an experimental analysis that compares the effectiveness of these heuristics working individually, and cooperating within an A-team framework. Additionally for calibration purposes we compare these results against feasible solutions derived using integer programming formulations. Since an important consideration in a real application is the computation time required to generate solutions we also compare the performance of such a cooperative problem solving strategy against traditional integer programming techniques.

The rest of the paper is organized as follows: Section 2 describes the cooperative problem solving strategy used to solve the bicriteria sparse multiple knapsack problem. A blackboard architecture based implementation is outlined. A brief overview of related AI and OR literature is provided. Section 3 introduces the multiple knapsack problem and a collection of heuristics to solve this problem. Section 4 presents results comparing the performance of the heuristics against the algorithm designed for each problem instance. A detailed analysis of the algorithms designed for each problem instance is provided.

2 Cooperative Problem Solving

Given an NP-hard optimization problem, it is difficult to design heuristic algorithms which exhibit uniformly superior performance over all problem instances. An alternate approach to tackle difficult problems is to organize a collection of heuristic algorithms so that they can cooperate with each other and uniformly exhibit superior performance which might not have been possible if they were used separately. Such an approach is especially attractive when the collection of heuristic algorithms vary in their performance over problem instances in an unpredictable way. Another ingredient required for cooperative problem solving is an architecture that facilitates cooperation between the heuristic algorithms and a control strategy that defines the rules of collaboration among heuristics.

In the following paragraphs we discuss in more detail the organization (i.e. the architecture and the control strategy) that we have used to build a cooperative problem solving team of heuristics for the multiple knapsack problem. We will also

discuss in some detail the collection of heuristics that have been used to build the cooperative problem solving team.

2.1 The Asynchronous Teams Architecture

An asynchronous team or A-Team [TdSM93] is an architecture that facilitates multiple heuristics to work together on a common problem. Cooperation between heuristics is allowed through a shared population of candidate solutions. Figure 1 provides a schematic of this architecture. The architecture is similar to a blackboard system in that the solutions are posted onto a blackboard which is shared by all the heuristic algorithms. Each heuristic has access to the entire population of solutions and can choose an appropriate (partial) solution to work on.

The heuristics that are used in this architecture are usually classified into three categories based as follows:

- *Constructors* are heuristics which are used to create initial solutions.
- *Improvers* are heuristics which take existing partial solutions from the population and modify them to produce a new solution. The criteria used to decide whether a solution is added to the population depends on the choice of the control strategy. In purely hill climbing approaches (such as genetic algorithms) only solutions which are non-dominated would be added to the population. In variants such as simulated annealing we might allow the entry of dominated solutions into the population with the expectation that they might allow for better solutions to be created later on.
- *Destroyers* are heuristics that remove redundant solutions from the population with the intent of managing the size of the population. The determination of whether a solution is redundant is difficult and usually a destroyer is designed to retain a redundant solution with a non-zero probability. Note that destroyers are typically used with control strategies that allow for the inclusion of dominated solutions into the population.

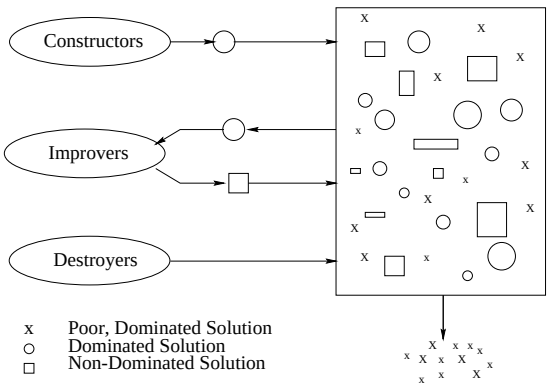


Figure 1: Schematic of an A-Team architecture

The typical approach for generating solutions using this architecture involves creating an initial population of solutions using all the constructor heuristics and subsequently evolving the population of solutions by repeated application of the improver and destroyer heuristics. At any time the set of non-dominated solutions constitute a Pareto-frontier and provide a set of non-dominated solutions to the problem. The control strategy in this solution approach prescribes the rules of collaboration between the heuristics. The control strategy specifies two rules:

- The first rule specifies how an improver heuristic picks a solution from the population to improve, and
- the second rule specifies the criteria for incorporating a new solution created by an improver heuristic into the population.

In our implementation we used a stochastic hill climbing approach for the control strategy. A stochastic hill climbing algorithm searches a space S with the aim of finding a state with optimal properties. The algorithm does this by making successive improvements to the current state $\rho \in S$. In the context of this paper, the state ρ corresponds to a population of solutions to the bicriteria multiple knapsack problem. The algorithm attempts to improve the current state ρ by making a transition to one of the neighbors τ of ρ in S. Within our A-Team implementation, this transition is made by randomly picking an improver heuristic and then randomly picking a solution from the population for the improver to work on. If the new solution generated by the improver is non-dominated then it is added to the population which corresponds to the new state τ. Since the hill climbing approach does not allow non-improving moves, we do not explicitly need to worry about managing the size of the population. As a result our implementation of A-team for the bicriteria sparse multiple knapsack problem did not require destroyers.

2.2 Related Work

There are two strands of AI research relevant to this paper. The first strand is the work on optimal composition of real-time systems to perform complex planning tasks [ZR95]. Complex systems are constructed by composing larger systems from smaller reusable anytime modules which might solve subproblems of the actual task at hand. Related work argues for the use of asymptotic bounded optimal agents as a useful bases for constructing intelligent systems using production system architectures [RS93, Rus95]. The use of an economics approach to design computational portfolios for solving hard problem has also been suggested [HLH97]. This paper applies these conceptual ideas to solving hard optimization problems in the real context of the multiple knapsack problem. The main variation is the use of stochastic hill-climbing for coordinating the interaction among heuristics.

Genetic programming (GP) is another strand in AI which formulates the search for a solution to an optimization prob-

lem as automatic programming [Koz92, Koz94]. A population of programs is herded using a set of evolutionary operators such as reproduction, mutations, and crossovers. The approach in this paper although inspired by GP differs in two aspects: (i) The programs in our context is defined as a sequences of heuristics rather than as a S-expression, and (ii) the search is guided by a stochastic hill climbing aproach rather than using evolutionary operators [JW94].

The single objective versions of the bicriteria problem are slightly modified forms of two well-known problems in the OR literature.

If we consider the objective of maximizing total assigned weight alone, then the problem is a variation of the *multiple knapsack problem*, which we call the sparse multiple knapsack problem (SMK). For the objective of minimizing total waste alone, the problem reduces to a variation of the *variable-size bin packing problem*, which we refer to as the sparse bin packing problem (SBP).

In the classical multiple knapsack problem, any item can go into any knapsack, hence the bipartite graph representing the problem is complete, whereas we generalize the problem by allowing any bipartite graph. On the other hand the multiple knapsack problem has a more general objective function: there exists a positive profit p_j for assigning item j to any of the knapsacks and the objective function is to maximize the total profit of assigned items. In the application which motivated us, the profit of an assigned item can be assumed to be proportional to the weight of the item, hence we maximize total assigned weight.

The multiple knapsack problem is known to be NP-hard in the strong sense [Kar72], [MT90]. The reduction from the 3-partition problem is still valid when the objective function coefficients equal to the weights of items, so any instance of SMK with a complete bipartite graph representation is also NP-hard in the strong sense. Thus, for the objective of maximizing total assigned weight, our problem is strongly NP-hard and there exists no fully polynomial time approximation scheme for SMK unless $P = NP$.

For the multiple knapsack problem, several exact and heuristic solution methods have been developed and tested in the literature (see Martello and Toth [MT90] for a survey). Exact solution methods consist of branch and bound, and cutting planes. The branch and bound methods use bounds based on either the Lagrangean relaxation[1] (Hung and Fisk [HF78]) or the surrogate relaxation [2] (Martello and Toth [MT80, MT81a]) of the problem. The cutting plane methods use minimal cover, (1,d)-configuration and multiple cover inequalities (Ferreira, Martin and Weismantel [FMW96]). Unfortunately, these exact solution methods cannot solve large instances arising in real applications in reasonable computation time. Heuristic methods include fast greedy algorithms followed by local exchange heuristics (Martello and Toth [MT81b]), as well as non-polynomial time approaches such as solving single knapsack problems successively (Martello and Toth [MT81a]), or obtaining a feasible solution from the surrogate relaxation (Hung and Fisk [HF78]).

Considering the objective of minimizing waste alone produces an ill-posed problem since the problem has a trivial solution of not assigning any items. Hence, we can consider a version in which we impose the condition that all items in N or a specified subset of N must be assigned and the goal is to use knapsacks with minimum total capacity. Then, the problem is a generalization of the variable-size bin packing problem, where we allow assignment restrictions in addition.

The bin packing problem is known to be NP-hard (i.e. [GJ79]), thus the more general problem SBP is NP-hard as well. The bin-packing problem has been extensively studied in the literature and it is one of the first problems for which efficient approximation algorithms have been developed. The recent survey by Cook, Garey and Johnson [CGJ97] covers worst and average case analyses for online and offline algorithms. A previous survey by the same authors [CGJ84] considers also some variations on the problem. Exact algorithms have been developed by Martello and Toth [MT89]. The *variable-size* bin packing problem has been studied by Friesen and Langston [FL86] who provide modifications of well-known bin packing heuristics such as the next fit, first fit and best fit heuristics.

3 Bicriteria Sparse Multiple Knapsack Problem

We are given a set of items $N = \{1, \ldots, n\}$ and a set of knapsacks $M = \{1, \ldots, m\}$. Each item $j \in N$ has a positive real weight w_j and each knapsack $i \in M$ has a positive real capacity c_i associated with it. In addition, for each item $j \in N$ a set $A_j \subseteq M$ of knapsacks that can hold item j is specified. Although A_j's suffice to represent the assignment restrictions, for convenience we also specify for each knapsack $i \in M$, the set $B_i \subseteq N$ of items that can be assigned to the knapsack.

The goal is to find an assignment of items to the knapsacks. That is, for each knapsack $i \in M$, we need to choose a subset S_i of items in N to be assigned to knapsack i, such that:

(1) All S_i's are disjoint. (Each item is assigned to at most one knapsack.)

(2) Each S_i is a subset of B_i, for $i = 1, \ldots, m$. (Assignment restrictions are satisfied.)

(3) $\sum_{j \in S_i} w_j \leq c_i$, for $i = 1, \ldots, m$. (Total weight of items assigned to a knapsack does not exceed the capacity of the knapsack.)

(4) $\sum_{i \in M} \sum_{j \in S_i} w_j$ is maximized. (Total weight of items assigned is maximized.)

[1] relaxing the assignment constraints decomposes the problem into m single knapsack problems.

[2] getting a linear combination of the capacity constraints results in a single knapsack problem.

(5) $\sum_{i \in I}(c_i - \sum_{j \in S_i} w_j)$ is minimized, where $I \subseteq M$ denotes the set of indices of non_empty S_i. (Total waste due to the unused portion of each utilized knapsack is minimized.)

We refer to this problem as the bicriteria sparse multiple knapsack problem (BSMK) [SKM97].

Note that the assignment restrictions can also be represented by a bipartite graph, where the two disjoint node sets of the graph correspond to the sets N and M. Let $G = (V, E)$ be the corresponding bipartite graph with $V = N \cup M$. Then, there exist an edge $(i, j) \in E$ between nodes i and j if and only if $j \in B_i$. With this representation the sparsity of the problem refers to the edge sparsity of the bipartite graph G. The *bicriteria* problem is more relevant for sparser problems because for more constrained problems, a solution with maximum assigned weight does not necessarily have small waste.

3.1 Constructor Heuristics

The construction heuristics are mainly greedy heuristics with various item and knapsack selection rules, in addition to a couple of heuristics that round the LP relaxation solution of SMK. Most of these constructors aim at maximizing total assigned weight.

Simple Greedy Heuristics

These heuristics first sort the items in non-increasing order of weight and the knapsacks in non-decreasing order of capacity. There are two versions. In the first one, the next knapsack in the order, say i, is picked, and then the next item, in the order, say j, is picked. If item j is allowed to go into knapsack j, i.e. if $j \in B_i$, and w_j does not exceed the residual capacity of knapsack i, then item j is assigned to knapsack i. So, for each knapsack picked, as many items as possible are packed into it. In the second version, each item in the order is picked and assigned to the next feasible knapsack, if possible. Both of the heuristics have running time $O(n \log n + m \log m + nm)$. We call these heuristics *greedy-knapsack* and *greedy-item.*

Greedy Heuristics with Various Knapsack Selection Rules

In these heuristics the decision to pick the next knapsack depends on the assignments made upto that point. The heuristics sort the items in non-increasing order of weight, pick the next item j and then pick a knapsack $i \in A_j$ according to one of three rules. There are three versions based on picking a knapsack with 1) minimum residual capacity, 2) maximum residual capacity, or 3) minimum surplus demand, which are called *greedy-minrc*, *greedy-maxrc*, and *greedy-minsd*, respectively. Surplus demand of a knapsack i is the total weight of unassigned items in B_i minus the residual capacity of the knapsack.

Successive Assignment Heuristic

This is another greedy heuristic, where at each iteration a maximum weight bipartite matching (assignment) problem is solved on a bipartite graph in which edge (i, j) exists with weight w_j only if w_j does not exceed the residual capacity of knapsack i. Initially the bipartite graph G is used. The assignments given by the maximum weight bipartite matching solution are performed. Then, the bipartite graph is updated by deleting all assigned nodes, all edges (i, j) for which w_j exceeds the residual capacity of knapsack j, and nodes with degree zero. The heuristic is repeated until the graph has no remaining edge. This heuristic is called *successive-assign.*

Randomized Heuristics

The greedy heuristics can be modified randomly in order to break the pattern of greedy choices. Suppose item j is picked in any of the greedy heuristics. The item will be considered for assignment with a probability p_j. After running through all items, the heuristic is repeated with $p_j = 1, \forall j$ in order to assign the remaining items. There are two versions based on the choice of p_j. In the first one p_j is proportional to the weight of item j. That is, $p_j = (w_j/\text{average weight of an item}) * C$, where C is a constant factor, so that items with larger weight are more likley to be picked. We call this heuristic *random-weight*. In the second version, called *random-degree*, $p_j = 1 - (deg(j)/\text{average degree in } G) * C$, where $deg(j)$ is the degree of node j in G.

Heuristics Based on the LP Relaxation of SMK

These heuristics solve the LP relaxation of the problem for the single objective of maximizing total assigned weight and then construct a feasible solution by rounding the fractional LP solution.

The IP formulation of SMK is as follows.

$$
\begin{aligned}
\max \quad & \sum_{i \in M} \sum_{j \in B_i} w_j \, x_{ij} \\
\text{st} \quad & \\
& \sum_{j \in B_i} w_j \, x_{ij} \le c_i, \;\; i \in M \\
& \sum_{i \in A_j} x_{ij} \le 1, \;\; j \in N \\
& x_{ij} \in \{0, 1\}, \; i \in A_j, \; j \in N
\end{aligned}
$$

where the 0-1 variable x_{ij} denotes whether item j is assigned to knapsack i. The LP relaxation corresponds to relaxing integrality of these variables.

The relaxation can be solved efficiently by a maximum flow algorithm. The continuous problem reduces to a maximum flow problem on a directed graph constructed from the bigraph G as follows. Each edge (j, i) of G is directed from the item node j to the knapsack node i and is assigned capacity w_j. A source node s is connected to each item node j via an arc (s, j) with capacity w_j. In addition a sink node t is connected to each knapsack node i via an arc (i, t) with capacity c_i. Then, the maximum flow from s to t equals the LP relaxation value and the amount of flow on arc (j, i) divided by w_j gives the value of x_{ij}. Thus, if flow on (j, i) equals w_j, i.e. $x_{ij} = 1$, then item j is assigned to knapsack i.

If $0 < x_{ij} < 1$, the variable is said to be fractional (in the corresponding solution).

There are two versions of the heuristic. In the first one, the fractional variables are rounded down and the remaining items are assigned by a simple greedy heuristic. In the second version, the fractional variables are sorted in non-increasing order of their values, and for each fractional variable in the order, the assignment is done if it is feasible. Then, the remaining items are assigned greedily as in the first version. These two heuristics are called *lp-greedy* and *lp-round*.

3.2 Improver Heuristics

The improver heuristics are either local exchange heuristics which aim to improve both of the objectives, or heuristics which rearrange assigned items among knapsacks and unassign some items for the purpose of minimizing total waste. We provide a brief description of each heuristic.

Local Exchange Heuristics

These local exchange heuristics aim at improving both of the objectives, and are repeated until no more improvement occurs.

1. Exchange Items Assigned to Different Knapsacks

   Consider all pairs of items assigned to different knapsacks. Swap the two items, if the swap is feasible and allows an unassigned item to be assigned to one of the two knapsacks. If the exchange of items is performed, pick the knapsack whose residual capacity has just increased and repetitivly assign the item with maximum weight to it, if the assignment is feasible. This heuristic is called *exchange*.

2. Replace Assigned Items with Unassigned Items

   Replace an assigned item or a pair of assigned items, with a single unassigned item or a pair of unassigned items of larger weight. These heuristics are called *replace-single* and *replace-pair* based on whether a single item or a pair of items are replaced.

3. Rearrange

   Rearrange assigned items in different knapsacks to aggregate residual capacity into one knapsack and then use the aggregated capacity to assign new items. Swap two items assigned to two different knapsacks, if the exchange is feasible and the maximum residual capacity (over all knapsacks) increases. After all pairs of items have been considered, repeat assigning a new item with maximum weight to the knapsack with maximum residual capacity, if feasible. This heuristic is called *rearrange*.

Heuristics to Eliminate Waste

1. Empty Under-utilized Slabs:

   This is a randomized heuristic. Two parameters are picked randomly: minimum allowable utilization and maximum allowable percentage decrease in assigned weight. Cancel assignments in all slabs with utilization less than minimum allowable utilization, if the decrease in weight is less than the maximum allowable percentage. This heuristic is called *empty*.

2. Empty Slabs Randomly and Reassign Orders

   For all knapsacks that are utilized, empty the knapsack with probability (1 - utilization of the knapsack). Then, reassign items by the greedy-minrc heuristic. This heuristic is called *empty-and-reassign*.

3. Pack Again

   These are variable-sized bin packing heuristics. Cancel assignments in all knapsacks. Reassign originally assigned items by first fit decreasing or best fit decreasing heuristics (i.e. the greedy-item or greedy-minrc heuristics). These heuristics are called *pack-again-ffd*, and *pack-again-bfd*, respectively.

Heuristics to Increase Assigned Weight

Any of the constructor heuristics can be used as an improver to assign the remaining items, if feasible. In our implementations we used the simple greedy-item heuristic for this purpose and we refer to it as *assign-remaining*.

3.3 Data

We used a real data set from an inventory application problem in the Process Industry [KDTL98]. For the instances available to us, the number of items vary between 111 and 439, while the number of knapsacks is between 18-43. The sparsity of the problems are in the range 10% - 28%. Size and sparsity of these instances are summarized in Table 1. The difficulty in solving these instances is shown in [SKM97]. An integer programming approach is unable to solve these problems optimally in over 4 hours.

Data	n	m	sparsity %	Tot. Cap.	Tot. Weight
d1	439	24	27.7	641.85	4689.91
d2	111	35	12.8	1009.32	1770.81
d3	393	18	26.7	388.84	4276.53
d4	209	43	10.6	889.21	3528.04
d5	191	35	14.2	730.81	2885.49
d6	155	18	18.6	446.32	1509.22

Table 1: Information on real-life data. Sparsity denotes the edge density of the bipartite graph representation in percentage of the number of edges of a complete bipartite graph. The last two columns denote the total capacity of knapsacks and the total weight of items.

4 Results

A comparison of the solutions with maximum assigned weight generated by A-team implementation and individual

runs is provided in Table 2. The waste of these solutions are also given in the table. The quality of the solutions generated by the A-team implementation is significantly better than the ones generated by individual runs, especially in the waste objective. We see that a meta-level search to sequence the heuristics have been useful to decrease the waste of the solutions that have the maximum assigned weight.

Solutions with maximum value of (assigned weight - waste), that are generated by the A-team implementation and individual runs are given in Table 3. We see a significant improvement in (assigned weight - waste) of the solutions generated by the A-team and individual heuristics, especially for the instances d2 and d4.

4.1 Analysis of Designed Algorithms

By examining the non-dominated solutions on the Pareto frontier (shown for data d4 in Figure 2) we can identify the heuristics and the sequence in which they were applied to yield a particular solution. Naturally the question arises whether it is really necessary to randomize the sequence in which these heuristics act on each others solutions. If it is possible to identify one or more sequences (or concatenation of heuristics) which yield the Pareto-frontier for all the problem instances then we might abandon a stochastic control strategy for constructing these solutions. In this section we show that the concatenation of heuristics used to generate the Pareto-frontier varies significantly by problem instance. This illustrates the need to tailor the solution strategy by problem instance which is automated by the stochastic control strategy adopted in this paper.

For each non-dominated solution generated by the A-team implementation, we traced the heuristcs whose output solutions were used to obtain the non-dominated solution. We can conclude that some heuristics were more effective and were repeated more. Nevertheless, we observed no regular patterns in the sequence of agents called across different problem instances. This motivates the use of an A-team approach for the concatenation of heuristics as opposed to identifying some effective patterns and using these patterns instead.

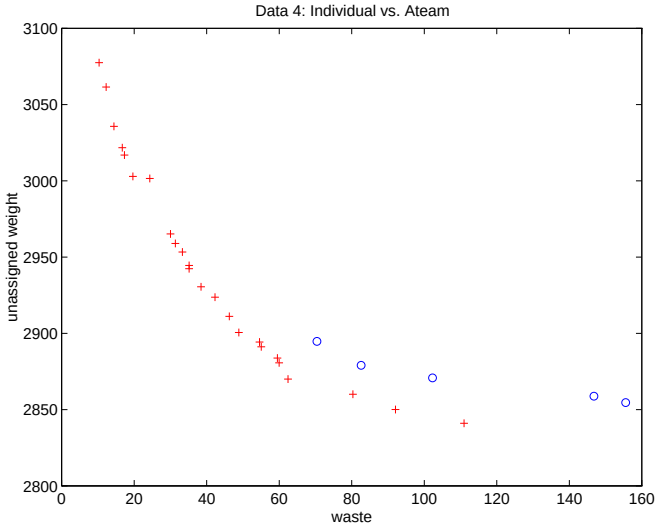
For the A-team implementation, the total number of occurrences of each heuristic which participates in the generation of non-dominated solutions (for each problem instance) is given in Table 4. We see that lp-round has been the most frequently used constructor. Local exchange heuristics replace-single and replace-pair, and waste reduction heuristics empty and empty-and-reassign were the most effective improvers in generating non-dominated solutions. We also give the heuristics that output the non-dominated solutions of the individual runs in Table 5 for comparison purposes. The statistics presented in these tables clearly indicate that the patterns used for constructing the Pareto-frontier varied significantly across problem instances. Figures 2 illustrates the construction of the non-dominated solutions for data d4 generated by the A-team implementation.

Data		AW	Ratio	Waste	Waste %	Cpu Time
d1	I	636.60	0.9952	5.25	0.82	4399.06
	II	617.69	0.9656	24.16	3.76	37.81
	III	601.18	0.9398	40.67	6.24	29622.41
d2	I	470.98	0.9975	108.24	18.69	70.01
	II	470.32	0.9961	230.50	32.89	1.48
	III	472.14	1.0000	110.04	18.90	8236.05
d3	I	383.20	0.9954	5.64	1.45	9834.42
	II	382.33	0.9931	6.52	1.68	72.32
	III	366.10	0.9509	22.74	5.85	46009.14
d4	I	686.99	0.9990	110.97	13.91	318.17
	II	673.39	0.9793	155.54	18.76	3.44
	III	686.99	0.9990	140.68	17.00	39923.20
d5	I	592.33	0.9899	96.92	14.06	183.70
	II	590.23	0.9864	99.02	14.37	3.06
	III	591.33	0.9882	97.92	14.21	39360.16
d6	I	406.12	0.9574	30.95	7.08	76.46
	II	402.92	0.9498	34.15	7.81	1.63
	III	402.97	0.9500	34.10	7.80	46570.32

Table 2: A comparison of the solution with *maximum assigned weight* obtained by I) the A-team implementation, II) individual runs of all heuristics, and III) branch-and-cut. AW is assigned weight. Ratio is the ratio of AW to the best available bound for the assigned weight objective. Waste % is the ratio of the unused capacity to the total capacity of utilized knapsacks in percentage. Cpu time is given in seconds.

Data		AW-Waste	Ratio	Waste %	Cpu Time
d1	I	631.35	0.99	0.82	4399
	II	593.53	0.93	3.76	38
	III	565.62	0.88	5.94	44543
d2	I	396.72	0.98	10.68	70
	II	326.48	0.81	3.33	2
	III	398.86	0.99	13.32	15681
d3	I	377.56	0.99	1.45	9834
	II	375.81	0.99	1.68	72
	III	354.36	0.93	4.43	28723
d4	I	595.55	0.98	8.67	318
	II	566.41	0.93	11.29	4
	III	605.34	0.99	8.58	24942
d5	I	497.14	0.97	12.54	183
	II	491.21	0.96	14.37	3
	III	465.90	0.91	14.93	23117
d6	I	375.17	0.91	7.08	76
	II	368.70	0.89	7.81	2
	III	365.83	0.88	8.15	18203

Table 3: A comparison of the solution with *maximum (assigned weight - waste)* obtained by I) the A-team implementation, II) individual runs of all heuristics, and III) branch-and-cut. The ratio is obtained using the best available upper bound for maximizing assigned weight minus waste. Cpu time is given in seconds.



Data	Constructors						
	LPR	GAR	GIR	GSD	GK	SA	RD
d1	1		1				
d2	1	1					
d3				1			
d4	1					1	
d5	1	1					1
d6	1		1		1		
Total	5	2	2	1	1	1	1

Data	Improvers							
	RS	RP	EM	EMR	EX	AR	PF	PB
d1	4	2						1
d2	13	8	14	3	2	1	1	
d3	2	3	2			1		
d4	15	5	26	5	3	1	1	
d5	12	7	27	7	2	2	1	
d6	3	5	6	1	1	1		
Total	49	30	75	16	8	6	3	1

Table 4: The frequency of the heuristics that yielded non-dominated solutions of the A-team implementation. Note that the heuristics that were never used do not exist in the table.

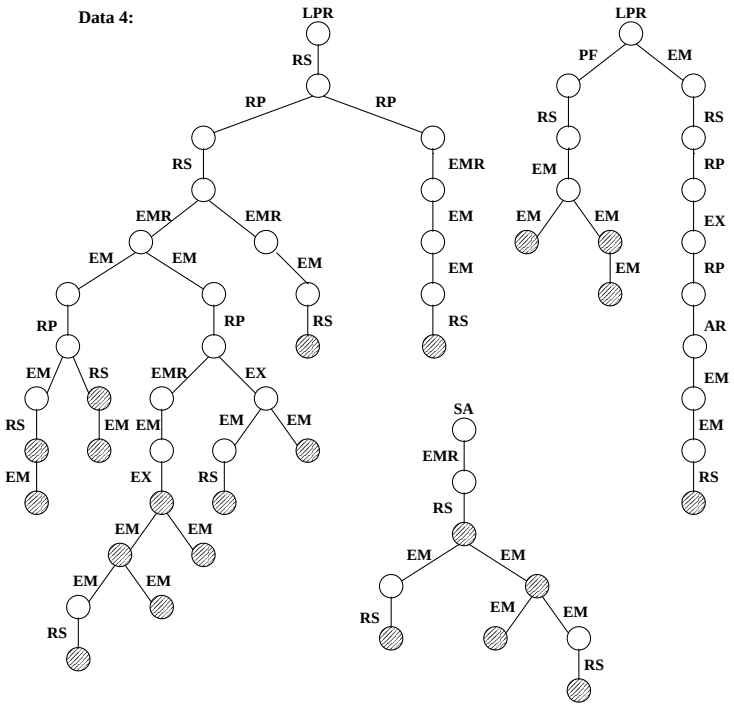


Figure 3: Trace of non-dominated solutions generated by the A-team implementation for data set 4.

Bibliography

[CGJ84] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packing: An updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithm Design for Computer System Design*, pages 49 – 106. Springer-Verlag, Wien, 1984.

[CGJ97] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packing: A survey. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 46 – 93. PWS Publishing Company, Boston, 1997.

[FL86] D.K. Friesen and M.A. Langston. Variable sized bin packing. *SIAM J. Computing*, 15:222 – 230, 1986.

[FMW96] C.E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM J. Optimization*, 6(3):858 – 877, 1996.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractibility:A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, 1979.

[HF78] M.S. Hung and J.C. Fisk. An algorithm for 0-1 multiple knapsack problems. *Naval Res. Logist. Quarterly*, 24:571–579, 1978.

Data	Constructors							
	LPG	GAR	GIR	GSD	GK	GI	RD	RW
d1							1	
d2		1	1		1	1		
d3								
d4				1				1
d5								
d6	1							
Total	1	1	1	1	1	1	1	1

Data	Improvers		
	RS	EM	EMR
d1	1		
d2	1	1	
d3	1		
d4	1	1	1
d5	1		
d6	1		
Total	5	2	1

Table 5: The heuristics that output non-dominated solutions of the individual runs. Note that for data d2, the solutions output by GI, GK and GIR heuristics are the same, and there exists a total of 4 non-dominated solutions.

[HLH97] B.A. Huberman, R.M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275:51–54, 1997.

[IK97] I.F. Imam and Y. Kodratoff. Intelligent adaptive agents. *AI Magazine*, 18(3):75–80, Fall 1997.

[JW94] A. Juels and M. Wattenberg. Stochastic hill-climbing as a baseline method for evaluating genetic algorithms. Technical Report CSD94-834, Dept. of Computer Science, University of California at Berkeley, 1994.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85 – 103. Plenum Press, New York, 1972.

[KDTL98] J. Kalagnanam, M. Dawande, M. Trumbo, and H. S. Lee. The surplus inventory matching problem in the process industry. Technical Report RC21071, IBM T.J. Watson Research Center, 1998.

[Koz92] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Koz94] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[MARW97] S. Murthy, R. Akkiraju, J. Rachlin, and F. Wu. Agent-based cooperative scheduling. pages 112–117. Constraints and Agents, AAAI-97 Workshop, 1997.

[MT80] S. Martello and P. Toth. Solution of the zero-one multiple knapsack problem. *Euro. J. Oper. Res.*, 4:322–329, 1980.

[MT81a] S. Martello and P. Toth. A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Math.*, 3:275–288, 1981.

[MT81b] S. Martello and P. Toth. Heuristic algorithms for the multiple knapsack problem. *Computing*, 27:93–112, 1981.

[MT89] S. Martello and P. Toth. *Knapsack Problems*. John Wiley and Sons,Ltd., New York, 1989.

[MT90] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Math.*, 28:59–70, 1990.

[RS93] S. J. Russel and D. Subramanian. Provably bounded-optimal agents. *J. of Artificial Intelligence Research*, 1:1–36, 1993.

[Rus95] S.J. Russel. Rationality and intelligence. pages 950–957. IJCAI-95, 1995.

[SKM97] F.S. Salman, J. Kalagnanam, and S. Murthy. Heuristics for solving the bicriteria sparse multiple knapsack problem. Technical Report RC 21059, IBM T.J. Watson Research Center, 1997.

[TdS93] S.N. Talukdar and P. de Souza. Asynchronous organizations for multi-algorithm problems. pages 286–293. Proceeding of 8th SIGAPP Symposium on Applied Computings, Feb. 1993.

[TdSM93] S.N. Talukdar, P. de Souza, and S. Murthy. Organizations for computer-based agents. *Engineering Intelligent Systems*, 1(2):–, 1993.

[ZR95] S. Zilberstein and S.J. Russel. Optimal composition of real-time systems. *Artificial Intelligence*, 79(2), 1995.