ELSEVIER

Information and Software Technology 43 (2001) 883–890

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

The next release problem

A.J. Bagnall, V.J. Rayward-Smith, I.M. Whittley

Computer Science Sector, School of Information Systems, University of East Anglia, Norwich NR4 7TJ, UK

Abstract

Companies developing and maintaining complex software systems need to determine the features that should be added to their system as part of the next release. They will wish to select these features to ensure the demands of their client base are satisfied as much as possible while at the same time ensuring that they themselves have the resources to undertake the necessary development. This situation is modelled in this paper and the problem of selecting an optimal next release is shown to be NP-hard. The use of various modern heuristics to find a high quality but possibly suboptimal solution is described. Comparative studies of these heuristics are given for various test cases.

Keywords: Next release; Software systems; Software enhancements

1. Introduction

A problem facing any company involved in the development and maintenance of large, complex software systems sold to a range of diverse customers is that of determining what should be in the next release of the software. In general, the company is faced with

- demands from their customers for a wide range of software enhancements,
- a situation where some enhancements will require (one or more) prerequisite enhancements,
- customers whose value to the company differs so that the requirements of favoured customers will be viewed as having more importance than those of less favoured customers,
- requirements that will take widely differing amounts of time and effort to meet.

The challenge for the company is to select a set of requirements that is deliverable within their own budget and which meets the demands of their (important) customers. Making an incorrect decision can prove a serious mistake. Good customers can be lost if their requirements are not met; the company itself may go over budget or even not deliver on time if costs are not controlled.

The motivation behind formulating the next release problem was a presentation by representatives of a large telecommunications company at an informal Workshop of the Software Engineering with Metaheuristic Innovative Algorithms (SEMINAL) Network [2] which identified it asagenuine industry problem.

2. The model

Let R denote all the possible software enhancements of the current system. Each customer, i, will have a set of requirements, $R_i \subseteq R$, and a weight, $w_i \in Z^+$, which is a measure of the customer's importance to the company. Associated with the set, R, is a directed, acyclic graph $G = (R, E)$, where

$(r, r') \in E$ iff r is a prerequisite of r'.

G is not only acyclic, it is also transitive since

$(r, r') \in E \& (r', r'') \in E \Rightarrow (r, r'') \in E.$

If the company decides to satisfy customer i's requirements, it must not only develop R_i but also develop

$\text{parents}(R_i) = \{r \in R | (r, r') \in E, r' \in R_i\}.$

Note that since G is transitive, the set

$\hat{R}_i = R_i \cup \text{parents}(R_i)$

includes all the requirements for which there is a path in G of length ≥ 0 terminating in a vertex in R_i. Thus, $\hat{R}_i$ comprises all the software enhancements which have to be developed in order to satisfy customer i. Each $r \in R$ has an associated cost, $\text{cost}(r) \in Z^+$, which is the cost of developing that enhancement assuming that any prerequisite is in

0950-5849/01/$ - see front matter
PII: S0950-5849(01)00194-X

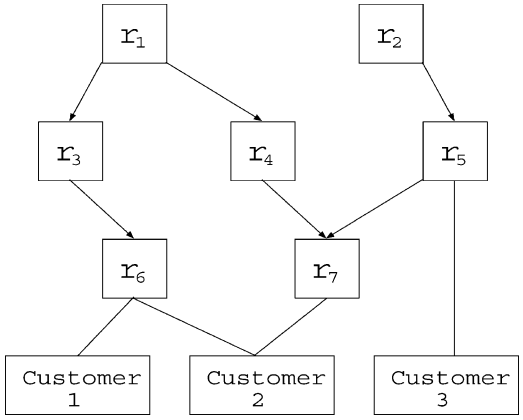

Fig. 1. An illustration of the structure of the example next release problem given in Section 2.

place. If $R' \subseteq R$, then we define

$$\text{cost}(R') = \sum \{\text{cost}(r) | r \in R'\}.$$

Assuming the company has n customers, its task is to find a subset of customers, $S \subseteq \{1, 2, \ldots, n\}$, whose requirements are to be satisfied. The cost of satisfying the customers in S is

$$\text{cost}\left(\bigcup_{i \in S} \hat{R}_i\right).$$

The problem faced by the company is thus to find a subset $S \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S} w_i \text{ is maximized}$$

subject to

$$\text{cost}\left(\bigcup_{i \in S} \hat{R}_i\right) \leq B$$

for some bound, $B \in Z^+$. A simple example NRP is as follows.

$$R = \{r_1, r_2, \ldots, r_7\}, \quad c_1 = 10, c_2 = 6, c_3 = 7, c_4 = 1, c_5 = 4, c_6 = 6, c_7 = 1,$$

$$R_1 = \{r_6\}, \quad R_2 = \{r_6, r_7\}, \quad R_3 = \{r_5\},$$

$$E = \{(1,3), (1,4), (1,6), (1,7), (2,5), (2,7), (3,6), (4,7), (5,7)\},$$

$$\hat{R}_1 = \{r_1, r_2, r_6\}, \quad \hat{R}_2 = \{r_1, r_3, r_4, r_6, r_7\}, \quad \hat{R}_3 = \{r_2, r_5\},$$

$$w_1 = 50, \quad w_2 = 60, \quad w_3 = 70.$$

A diagrammatical representation of the structure of this NRP problem is shown in Fig. 1.

Since $\hat{A} \cup \hat{B} = \widehat{A \cup B}$, we note that an alternative expression for

$$\text{cost}\left(\bigcup_{i \in S} \hat{R}_i\right)$$

is

$$\text{cost}(\hat{X}) = \text{cost}(X) + \text{cost}(\text{parents}(X) \backslash X),$$

where

$$X = \bigcup_{i \in S} R_i.$$

In the special case where no requirement has any prerequisites, $E = \varnothing$, we say that the next release problem is basic. In this case, $R_i = \hat{R}_i$ for any subset, so the company simply wishes to find an S to maximize $\sum_{i \in S} w_i$ subject to

$$\text{cost}\left(\bigcup_{i \in S} R_i\right) \leq B.$$

Clearly, any next release problem can be formulated as a basic next release problem by simply preprocessing each R_i and resetting it to $R_i \cup \text{parents}(R_i)$:

A more restrictive special case arises when customers' requirements are not only basic but also independent, i.e. we also have

$$\hat{R}_i \cap \hat{R}_j = R_i \cap R_j = \varnothing \text{ whenever } i \neq j.$$

In this special case,

$$\text{cost}\left(\bigcup_{i \in S} R_i\right) = \sum_{i \in S} \text{cost}(R_i).$$

So in the independent, basic case, the company wishes to maximize

$$\sum_{i \in S} w_i$$

subject to

$$\sum_{i \in S} c_i \leq B,$$

where $c_i = \text{cost}(R_i)$.

We thus need to find a Boolean variable, $x_i \in \{0, 1\}$, for each customer i, $1 \leq i \leq n$, such that

$$\sum_{i=1}^{n} w_i x_i$$

is maximized, subject to

$$\sum_{i=1}^{n} c_i x_i \leq B.$$

$x_i = 1$ if the customer, i, will have its requirements satisfied and $x_i = 0$ otherwise.

The above problem is the well-known 0/1 knapsack problem and its associated decision problem is known to

be NP-complete [6]. Hence, we have established the following result.

Theorem 1. *The next release problem is NP-hard even when the problem is basic and customer requirements are independent.*

The practical significance of this result is that (unless P = NP, which is highly unlikely), the next release problem cannot be solved by a polynomial time algorithm. Although it is possible to find the optimal solution using enumerative algorithms for small cases, as the number of customers increases so the problem will become increasingly intractable.

Rather than find a provably optimal solution, we will aim to exploit a range of (modern) heuristic algorithms which purport to find good quality, but possibly suboptimal, solutions.

We will assume the problem is in its most general form, i.e. we assume that we are given customer requirements that will have prerequisites and will not be independent. Thus our input will comprise a number, n, of customers and for each $1 \leq i \leq n$, a set of requirements, $R_i \subseteq R$. We will also have a directed, acyclic graph, $G = (R, E)$, representing prerequisites. We assume G is transitive, but if not, we can easily compute its transitive closure using Warshall's algorithm [11]. If the cardinality of R is m, the digraph, G, will be represented as an $m \times m$ Boolean matrix, M, where

$$M[i,j] = T \text{ iff } (r_i, r_j) \in E.$$

Then to compute parents(R_k), for any $R_k \subseteq R$, we compute a Boolean array, P_k, where

$$P_k[i] = T \text{ iff } r_i \in \text{parents}(R_k),$$

i.e. to compute P_k, we simply 'or' together columns of the matrix, M, corresponding to the elements of R_k. This is an $O(|R_k|m)$ operation.

Thus, if we need to evaluate the cost of meeting the requirements of p customers, $1 \leq p \leq n$, we can find this cost in $O(pqm)$ time where q is the largest number of requirements of any of the customers. Clearly, $p \leq n$ and $q \leq m$ so this evaluation is $O(nm^2)$ but this is very much aworst case bound and, in practice, evaluation will be fast. Because of this, it is appropriate to consider the use of algorithms such asgenetic algorithms, simulated annealing or tabu search tosolve the next release problem.

3. Algorithms

Three general approaches have been developed to tackle the next release problem (NRP). The first uses exact techniques to solve a linear programming relaxation of the problem to allow upper bounds to be calculated. This is then tightened into a full integer programming model and a generic branch and bound algorithm is applied. In the second, a set of simple greedy algorithms is developed allowing for the fast generation of lower bounds. The best of these techniques is then enhanced along GRASP lines to improve the values obtained. Finally, a set of standard local search techniques is applied using a simple neighbourhood structure, with the aim of generating near optimal solutions without using excessive amounts of computing time.

The problem set itself consists of five randomly generated problems. The smallest problem (NRP1) has 100 customers and 140 tasks, while the largest (NRP5) has 500 customers and 3250 tasks. Further details can be found in Appendix A. For each problem, the total resources required to satisfy all customers was calculated and experiments were performed using a bound, B, of 30, 50 and 70% of this total.

3.1. Exact techniques

3.1.1. IP formulation

The first step required in adapting the knapsack formulation to handle the general NRP is to separate the set of customers from the set of enhancements.

Let $x_1, x_2, \ldots x_m$ and $y_1, y_2, \ldots y_n$ be 0/1 variables representing the set of requirements and the set of customers, respectively. With w_i and c_i retaining their meaning we now have the following basic formulation:

Maximize

$$\sum_{i=1}^{n} w_i y_i$$

subject to

$$\sum_{i=1}^{m} c_i x_i \leq B,$$

$$x_i, y_j \in \{0,1\}\ 1 \leq i \leq m,\ 1 \leq j \leq n.$$

To complete the model requires the introduction of two more sets of constraints: one to model the dependency graph and a second to model the customer requirements.

Consider an edge, $(i,j) \in E$. The presence of this edge in the dependency graph species that if enhancement j is required, enhancement i must also be implemented. This relationship can be modelled as

$$x_j = 1 \Rightarrow x_i = 1$$

or, alternatively, in a form more suitable for a standard linear model:

$$x_i \geq x_j.$$

A constraint of this form is required for every edge, e, in our graph.

The customer requirements can be handled similarly adding, as they do, simply another set of dependencies into our model.

Table 1
Results for NRP using an MIPS solver (Results in bold denote an optimum solution, other results provide an upper bound on the maximum.)

Problem	Proportion of resources	Solution found
NRP1	0.3	**704**
	0.5	**1151**
	0.7	**1645**
NRP2	0.3	**2921**
	0.5	5024
	0.7	9001
NRP3	0.3	4803
	0.5	7454
	0.7	10 106
NRP4	0.3	4167
	0.5	6790
	0.7	9413
NRP5	0.3	**9601**
	0.5	**15 118**
	0.7	**20 726**

$$x_i \geq y_j$$

for all requirements, r_i, requested by each customer, j.

With these constraints in place, our full model can now be given as: maximize

$$\sum_{i=1}^{m} w_i y_i$$

subject to

$$\sum_{i=1}^{n} c_i x_i \leq B,$$

$$x_i \geq x_j \ \ \forall (r_i, r_j) \in E,$$

$x_i \geq y_j \ \ \forall i, j$ where r_i is requested by customer j.

$$x_i y_j \in \{0, 1\} \ \ 1 \leq i \leq m, \ 1 \leq j \leq n.$$

This is an integer linear program (ILP) because of the integrality constraints,$x_i y_j \in \{0, 1\}$. If these are relaxed to

$$0 \leq x_i \leq 1 \ \forall 1 \leq i \leq m,$$

$$0 \leq y_i \leq 1 \ \forall 1 \leq j \leq n,$$

the problem becomes a linear program which is much easier to solve. Solutions to this LP relaxation will provide an upper bound to the solution of the ILP.

3.1.2. Results

Using CPLEX [1], upper bounds were calculated for all problem instances by solving the LP relaxation of each problem with no run taking longer than 2 min. The integrality constraints were then reintroduced and further experimentation performed. The results are presented in Table 1.

- On the first problem, NRP1, optimal solutions were found with little difficulty for all cases, with the default CPLEX settings in place.
- When applied to the second problem, NRP2, the performance dropped off dramatically, as expected. On the most constrained instance, NRP2 03, it took CPLEX 40 min to reach the best solution and another 10 h before optimality could be guaranteed. The performance deteriorated further on NRP 05 with the run having to be terminated after 20 h.
- Problems 3 and 4, having similar structure to Problem 2, but being considerably larger in size were not attempted.
- Problem 5, whilst having the largest number of customers, provided only a relatively small number of enhancements for those customers to choose between. The structure of the problem proved very simple for

Table 2
Greedy algorithm results: grasp results are the best obtained from 10 runs

		Maximum profit	Minimum cost	Maximum ratio	GRASP(3/5/10)
NRP1	0.3	516	589	672	672/685/693
	0.5	845	968	1088	1107/1106/1094
	0.7	1291	1336	1484	1553/1602/1602
NRP2	0.3	1403	2268	2587	2657/2680/2652
	0.5	2835	3741	4242	4447/4447/4477
	0.7	5718	6099	6436	6849/6857/6867
NRP3	0.3	2736	4182	4553	4553/4553/4551
	0.5	5505	6437	6939	6996/7021/6988
	0.7	9133	9371	9846	9919/9916/9895
NRP4	0.3	1274	2794	3085	3121/3125/3089
	0.5	3355	4906	5200	5276/5282/5276
	0.7	6758	7791	8112	8249/8240/8232
NRP5	0.3	5251	7661	8588	8621/8681/8613
	0.5	10 673	12 307	13 316	13 570/13 628/13 676
	0.7	18 263	17 362	18 930	19 360/19 382/19 416

Table 3
Best results from 10 runs using first found, steepest assent and sampling hill climbers

Problem		FFHC	SAHC	SHC
NRP1	0.3	472	540	704
	0.5	925	874	1094
	0.7	1407	1421	1537
NRP2	0.3	1069	2081	2671
	0.5	2311	3875	4290
	0.7	4829	6734	6771
NRP3	0.3	2634	3634	4290
	0.5	4656	5859	6766
	0.7	8069	8491	9691
NRP4	0.3	1569	2502	2995
	0.5	3196	4355	5075
	0.7	5522	6746	7837
NRP5	0.3	4358	5251	7237
	0.5	9361	11 657	11 914
	0.7	17 543	18 515	19 005

CPLEX, with the root rounding heuristic able to fix over 90% of the variables before the branch-and-bound phase began. By also introducing a good lower bound, it was possible to obtain results for this problem using only a few minutes of computation time.

3.2. Greedy algorithms

Three simple greedy algorithms have been developed and applied to the problem set. These algorithms are constructive in nature and start with an empty set of satisfied customers. At each iteration, a customer is added to the set until no further additions can be made without exceeding the resource bound. The choice of which customer to select at each iteration is guided by a set of simple metrics.

The first such algorithm simply selects the customer with the highest profit rating out of those remaining. If enough resources remain that customer is added to the set, if not they are discarded and the next customer is considered. The second algorithm takes the approach of selecting the customer with the minimum resource requirement. After each selection, the list of candidates has to be re-ordered due to the inter-relationships between customer demands. The final approach is a combination of the previous two, ordering the customers on the ratio of their profit rating to the amount of resources they require.

The algorithms were each applied to all five problems with the results as given in Table 2. Of the three, the maximum ratio algorithm returns the best results, but comparison with the results given in Section 3.1 shows that the bounds are not truly competitive.

In an effort to improve the values obtained, the Maximum Ratio algorithm was enhanced by using ideas from the greedy randomized adaptive search procedure literature (GRASP) (see, e.g.Ref. [3]). Rather than selecting the best customer to enter the solution at each iteration, the algorithm was adapted to choose randomly from amongst the top k customers on the candidate list. Results can be found in Table 2 for values of k set to 3, 5 and 10, with the best value returned from 10 independent runs being recorded. By utilizing a GRASP approach the results have been improved but with an increase in computing time.

3.3. Local search

Local search techniques are iterative in nature and rely on the definition of a solution *neighbourhood* — a set of solutions that can be reached by making small changes to the current solution. For the NRP, a customer-based neighbourhood was defined. This approach was taken because

1. it is simple to implement,
2. it has a simple objective function, f, where

$$f(S) = \sum_{i \in S} w_i - \lambda \min\left\{0, B - cost\left(\bigcup_{i \in S} \hat{R}_i\right)\right\}.$$

A task-based neighbourhood could also be considered, but that is a topic for further research.

If we let $N(S)$ represent the neighbourhood of a solution, S, a move, m, simply adds a customer to the solution or removes one from it to obtain a new solution $m(S) \in N(S)$. We define the benefit obtained from the move by $b(m, S) = f(m(S)) - f(S)$. Then, the move m, on S, will be called improving iff $b(m, S) > 0$.

The strategies we are going to examine in this document can all be seen to construct a path through the search space in the following manner:

$$S_0 \xrightarrow{m_1} S_1 \xrightarrow{m_2} \cdots \xrightarrow{m_n} S_p$$

where $(S_{i-1}) = S_i$ for $1 \leq i \leq p$. Details will now be given on two families of algorithms that fit into this scheme.

3.3.1. Hill climbers

Hill climbing algorithms work by moving from a starting solution to a local optima by performing only improving moves. Three such algorithms were implemented for the NRP:

- steepest ascent,
- first found,
- sampling.

In a pure hill climber, m, is an acceptable move on S only if it is an improving move. The choice of which improving move to take depends on the particular strategy chosen.

The first of the three algorithms performs a complete enumeration of the neighbourhood and takes the best improving move found. The second, as its name suggest,

Table 4
Simulated annealing results (if the entry is in bold, it indicates the known optimum is achieved. If it is in italics it indicates it is the best known solution but has not been proved optimal.)

Problem		Geometric SA		Lundy Mees SA		
		250	500	5×10^{-7}	1×10^{-7}	1×10^{-8}
NRP1	0.3	**704**	**704**	**704**	**704**	**704**
	0.5	**1151**	**1151**	**1151**	**1151**	**1151**
	0.7	**1645**	**1645**	**1645**	**1645**	**1645**
NRP2	0.3	2846	2904	2819	2888	2914
	0.5	4965	4958	4934	4983	5007
	0.7	7845	7890	7824	*7893*	7864
NRP3	0.3	4636	4608	4583	*4647*	4636
	0.5	7091	7156	7083	7177	*7217*
	0.7	9952	10 001	9992	10 021	10 032
NRP4	0.3	3100	3117	3090	3106	3153
	0.5	5338	5355	5295	5360	5394
	0.7	8236	8279	8245	8288	*8355*
NRP5	0.3	9247	9364	9188	9285	9456
	0.5	14 774	14 881	14 725	14 891	14 975
	0.7	20 600	20 526	20 440	20 576	20 603

accepts the first improving move it encounters. The final algorithm simply samples, randomly, a number of moves from the neighbourhood (25 in the experiments performed) and takes the best of these — again providing that it is strictly improving.

After a number of iterations, the search will terminate. In the case of the first two algorithms, this will occur at a local optima as defined by the neighbourhood; for the third this is not necessarily the case — expanding the size of the sample could have identified an improving move.

If the solution obtained at this point is not of a satisfactory quality, the usual option is to repeat the algorithm from different starting points with the hope of identifying different, possibly higher quality, local optima. With the third algorithm, however, we have experimented with continuing the search by taking the best move irrespective of whether or not it is improving. In the results given in Table 3, this is the process that was performed.

Each value is the best found from ten runs of each algorithm on the problem in question, each starting from a different randomly generated solution. In the case of the two deterministic algorithms, the natural termination provided by a local optima was used to signal the need for a restart whilst for the sampling hill climber 10,000 moves were performed with the solution reported being the best encountered during the search rather than the final solution. As the table shows, by continuing the search from the vicinity of a good solution an improved quality of solution can be found.

3.3.2. Simulated annealing

Simulated annealing was introduced as an optimization algorithm in a 1983 paper by Kirkpatrick et al. [10] based on an algorithm put forward 30 years previously by Metropolis et al. [9] to model the cooling of materials in a heat bath — a process known as annealing.

The algorithm itself is a simple extension to the stochastic hill climber presented in Section 3.3.1 and it is this simplicity, combined with good empirical and theoretical results, that has led to its wide acceptance by the optimization community.

The algorithm works by allowing non-improving moves to be made but in a controlled manner. After each move is evaluated, it is assigned a probability for acceptance. This probability depends upon the cost benefit of the move and also the current state of the search, modelled by a parameter known as the *temperature*, t. At the start, the temperature is high and the search can accept moves of even poor quality. As the search progresses and the temperature cools, the probability drops until in the final phase only improving moves are accepted. It should also be noted that the algorithm is of the *first found* variety, accepting the first satisfactory move, it encounters in the neighbourhood and that the moves are selected for consideration randomly.

The algorithm is traditionally presented as described above with the following acceptance function:

$$P(m(S) \text{ is accepted}) = \begin{cases} 1 & \text{if } b(m,S) > 0, \\ e^{b(m,S)/t} & \text{otherwise.} \end{cases}$$

With the acceptance function in place, all that is required is the specification of the *cooling schedule*. Two standard approaches are presented below:

- Geometric: $t_{i+1} = \alpha t_i$,
- Lundy and Mees [7] $t_{i+1} = t_i/(1 + \beta t_i)$.

where α and β are control parameters. Both algorithms also require an initial temperature to be specified. The geometric SA (GSA) requires a parameter specifying how long to remain at each temperature stage; for the Lundy and Mees cooling schedule, the temperature drops after each move.

In the results given in Table 4, the starting temperature was set to 100 and for the GSA the multiplier, α, was set to 0.9995. The parameters specified in the table are for the number of moves at each temperature stage for the GSA and the β value for the SA with Lundy and Mees cooling schedule. The first four columns of results present the best value from 10 runs on each problem; the final column gives the results from one long run of between 30 and 60 min on each problem.

Table 5 illustrates how an SA approach out performs both greedy algorithms and hill climbers, doing particularly well on the harder NRP3 and NRP4 problems.

4. Conclusions and further work

A variety of algorithms have been applied to the NRP with varying degrees of success. On the smallest of the problems exact techniques proved to be sufficient, while

Table 5
Results for the best greedy algorithm (Grasp 10), hill climber (SHC) and simulated annealing algorithm (Lundy Mees SA 1×10^{-8})

Problem		Upper Bound	Grasp 10	SHC	Lundy Mees SA 1×10^{-8}
NRP1	0.3	704	693	704	704
	0.5	1151	1094	1094	1151
	0.7	1645	1602	1537	1645
NRP2	0.3	2921	2652	2671	2914
	0.5	5024	4477	4290	5007
	0.7	9001	6867	6771	7864
NRP3	0.3	4803	4551	4290	4636
	0.5	7454	6988	6766	7217
	0.7	10 106	9895	9691	10 032
NRP4	0.3	4167	3089	2995	3153
	0.5	6790	5276	5075	5394
	0.7	9413	8232	7837	8355
NRP5	0.3	9601	8613	7237	9456
	0.5	15 118	13 676	11 914	14 975
	0.7	20 726	19 416	19 005	20 603

on the remainder, simulated annealing algorithms found the best solutions using only modest amounts of computing time.

In order to apply these results to a real world problem, software project managers must first formulate the problem. This involves the identification of possible requirements, the specification of the relationship between these requirements and a quantification of the cost of implementation (i.e. determine R, E and cost(r) $\forall\, r \in R$). A survey of the customer base should then be conducted to identify their preferences for the next release and an assessment of the importance of satisfying each customer is required (i.e. determine customer set S, R_i and w_i $\forall\, i \in S$.). Having completed the formulation the results presented in this paper suggest that problem should first be approached with a integer programming approach. However, on larger problems this may fail to yield an optimal solution in reasonable time, in which case a SA approach may be used to find a good quality, but possibly suboptimal, solution.

To rate the quality of the solutions themselves requires comparison with the optimal solution (if known) or the upper bounds. The SA approaches to within 1.5% of the optimal value for all problems with known optima. For the remaining problems, one can expect the performance to be similar. We can be sure that the SA solution provides a lower bound and the LP relaxation an upper bound on the optimal solution. However, the LP relaxation provides a poor UB in many cases.

Despite these results, there is scope for further development on both the heuristic and exact techniques.

On the heuristic side, there remains the option of utilising the task based neighbourhood scheme and also introducing techniques such as tabu search [4,5] and ant colony optimization [8]. Both use memory to guide a search with the former controlling a local search algorithm and the latter enhancing greedy algorithms in a manner similar to GRASP.

For the exact techniques, there is a lot of scope for taking advantage of the special structure of the problem itself and also for incorporating some of the many advances that have been successfully used to solve the knapsack problem.

Table A1
Problems: these problem instances can be found on www.sys.uea.ac.uk/~ imw/nrp/problems.html

	NRP 1	NRP 21	NRP 3
Nodes/level	20/40/80	20/40/80/160/320	250/500/750
Cost/node	1–5/2–8/5–10	1–5/2–7/3–9/4–10/5–15	1–5/2–8/5–10
Maximum children/node	8/2/0	8/6/4/2/0	8/2/0
Customers	100	500	500
Requests/customer	1–5	1–5	1–5
	NRP 4	NRP 5	
Nodes/level	250/500/750/1000/750	500/500/500	
Cost/node	1–5/2–7/3–9/4–10/5–15	1–3/2/3–5	
Maximum children/node		8/6/4/2/0	4/4/0
Customers	500	1000	
Requests/customer	1–5	1	

Appendix A. The problem sets

The problems were developed along a multi-level approach (see Fig. 1). Each level is specified by

- the number of nodes it contains,
- the number of children each node has,
- the cost of each node.

The number of nodes for each level was specified in advance, while the other parameters were selected randomly from within a given range. Details for each problems are given in Table A1.

References

[1] R.E. Bixby, M. Fenelon, X.Z. Gu, R. Rothberg, Mip: theory and practice closing the gap, System Modelling and Optimization Methods, Theory and Applications (2000) 19–49.
[2] N. Drakos, et al., The SEMINAL Network, http://homepages.gold.ac.uk/mark/seminal/.
[3] T.A. Feo, M.G.C. Resende, Greedy randomised adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.
[4] F. Glover, Tabu search part i, ORSA Journal on Computing (1989).
[5] F. Glover, Tabu search part ii, ORSA Journal on Computing (1990).
[6] R.M. Karp, Reducability among combinatorial problems, Complexity of Computer Computations (1972).
[7] M. Lundy, A. Mees, Convergence of an annealing algorithm, Mathematical Programming 34 (1986) 111–124.
[8] V. Maniezzo, M. Dorigo, N. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics — Part B (1996).
[9] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculation by fast computing machines, Journal of Chemical Physics 21 (1953) 1087–1091.
[10] C.D. Gellat, S. Kirkpatrick, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[11] S. Warshall, A theorem on Boolean matrices, JACM 9 (1) (1962) 11–12.