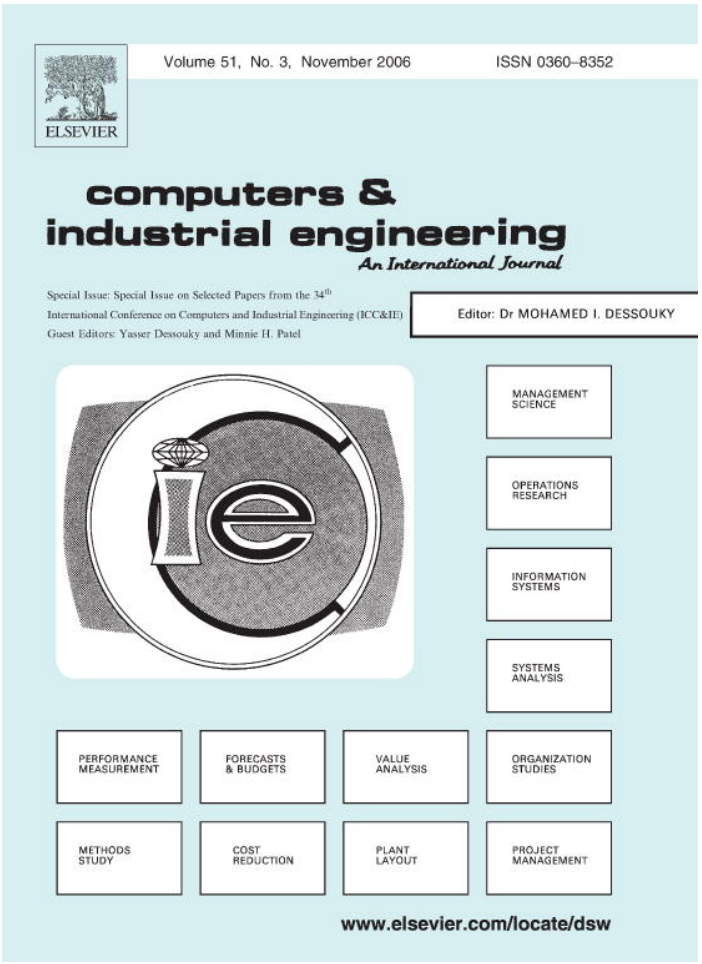

Available online at www.sciencedirect.com
ScienceDirect
Computers & Industrial Engineering 51 (2006) 362–374
computers & industrial engineering
www.elsevier.com/locate/dsw

On the effectiveness of incorporating randomness and memory into a multi-start metaheuristic with application to the Set Covering Problem

Guanghui Lan [a,*], Gail W. DePuy [b]

[a] School of Industrial and Systems Engineering, Georgia Institute of Technology, 765 Ferst Drive, NW, Atlanta, GA 30332-0205, USA
[b] Department of Industrial Engineering, University of Louisville, Louisville, KY 40292, USA

Available online 12 September 2006

Abstract

The construction of good starting solutions for multi-start local search heuristics is an important, yet not well-studied problem. In these heuristics, randomization methods are usually applied to explore new promising areas and memory mechanisms are incorporated with the main purpose of reinforcing good solutions. Under the template of a typical multi-start metaheuristic, Meta-RaPS (Meta-heuristic for Randomized Priority Search), this paper presents several randomization methods and memory mechanisms with a focus on comparing their effectiveness and analyzing their interaction effects. With the Set Covering Problem (SCP) as the application problem, it is found that these randomization methods work well for Meta-RaPS with an improvement phase while the memory mechanisms better the solution quality of the construction phase. The quality and efficiency of Meta-RaPS can be improved through the use of both memory mechanisms and randomization methods. This paper also discovers several efficient algorithms that maintain a good balance between randomness and memory and finds the optimal or best-known solutions for the 65 SCP test instances from the OR-library.

Keywords: Metaheuristic; Multi-start methods; Meta-RaPS; Memory; Set covering

1. Introduction

Multi-start heuristics can usually be characterized as iterative procedures consisting of two phases: the construction phase in which a solution is generated and the improvement phase in which the starting solution is typically (but not necessarily) improved by neighborhood search methods. The entire procedure is repeated a number of iterations or certain amount of time, and the best solution is then reported. As each iteration usually produces a local optimal solution, restarting the search helps to explore new areas in the solution space and overcome the local optimality. Randomness has been incorporated into most multi-start heuristics to

* Corresponding author.
E-mail address: glan@isye.gatech.edu (G. Lan).

0360-8352/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2006.08.002

construct diversified starting solutions. In the classical multi-start heuristics the starting solutions are generated completely randomly. As a well-known multi-start heuristic, GRASP (Greedy Randomized Adaptive Search Procedure) (Feo, Mauricio, & Resende, 1995) attempts to generate starting solution with good solution quality by introducing randomness systematically into greedy heuristics. Meta-RaPS (Meta-heuristic for Randomized Priority Search) is a more recent multi-start meta-heuristic developed by DePuy, Whitehouse, and Moraga (2002). It evolved from a heuristic designed by Arcus (1966) to solve the assembly line balancing problem – COMSOAL (Computer Method of Sequencing Operations for Assembly Lines). The theme of Meta-RaPS is to utilize randomness in a systematic fashion. As will be explained later, Meta-RaPS can also be viewed as a generalization of some other algorithms, such as GRASP, and is more flexible and more effective in many cases. In addition, several multi-start heuristics designed for particular problems that integrate some random factors can be found in the literature (Boese, Kahng, & Muddu, 1994; Patterson, Rolland, & Pirkul, 1999).

One way to improve the starting solutions is to incorporate a memory mechanism into the construction phase which tries to identify elements that are common to good solutions and utilize this information to produce better starting solutions in later iterations, such that the solution quality and/or efficiency of the overall algorithm can be improved. Fleurent and Glover (1999) found that incorporating adaptive memory can enhance the solution quality for the construction phase when applying GRASP for the Quadratic Assignment Problem (QAP). It should be noted that memory mechanisms are usually coupled with certain diversification elements to avoid quick convergence to the same local optimum. On the other hand, since an important principle for multi-start heuristics is to achieve proper diversity in the construction phase while the intensified search is applied in the improvement phase, the lack of memory is not as unreasonable as might be imagined as constructing independent starting solutions may be viewed as strategically sampling the solution space (Martí, 2003). Actually, the effectiveness of incorporating memory into the multi-start heuristics in which the intensified neighborhood search is applied has not been well studied.

This research investigates the effectiveness of a variety of randomization methods and memory mechanisms in Meta-RaPS and attempts to maximize the benefits from both methods. We first present methods to introduce random factors into the construction phase. Since Meta-RaPS is originally a memoryless metaheuristic in that it does not learn from previous iterations, we then devise a variety of memory mechanisms that include how to represent the information from the previous iterations and how to make use of these pieces of historical information. The major effort focuses on the experimental study to analyze the relative benefits from these randomization and memory methods and proposes algorithms that maintain a good balance between randomness and memory to improve the overall solution quality and efficiency.

Notice that memory mechanisms have also been incorporated into modern implementations of GRASP (Greedy Randomized Adaptive Search Procedure), a popular multi-start local search heuristic (Pitsoulis & Resende, 2002; Resende & Ribeiro, 2003, 2005). For example, in reactive GRASP, the parameter to control the construction of starting solutions is adjusted according to the quality of the previously generated solutions. A significant breakthrough in GRASP is to incorporate path-relinking as an additional phase each iteration after the local search, and/or as an post-optimization phase after all iterations of GRASP have been performed. The memory methods used in Meta-RaPS differ from those used in both reactive GRASP and GRASP with path-relinking in the following aspects. First, the memory methods presented in this paper, namely, priority rules with fitness and partial construction, are intrinsic in the construction phase to obtain good starting solutions. They are neither methods to adaptively adjust parameter settings nor extended post-optimization phases. More importantly, these memory methods in Meta-RaPS are expected to collaborate with several randomization techniques, which are also intrinsic in the construction phase to diversify the search, while reactive GRASP or GRASP with path-relinking only applies the intensification techniques. Third, Meta-RaPS gathers historical information pertaining to basic elements in a solution to evaluate their utility, while reactive GRASP and GRASP with path-relinking usually only seek information about solutions for neighborhood transition. In Section 5.4 we provide experimental results, which show that Meta-RaPS outperforms an implementation of reactive GRASP with path relinking. Nevertheless, we believe that the post-optimization phases or parameter-setting techniques in the modern GRASP may be incorporated into Meta-RaPS, which would involve the trade-off between solution quality and efficiency.

The Set Covering Problem (SCP) is used to evaluate the randomness and memory techniques developed in this paper. The SCP is a fundamental problem in operations research and usually described as the problem of covering the rows of an m-row, n-column, zero-one matrix (a_{ij}) by a subset of the columns at minimal cost. The formal binary integer program model of the SCP is defined as follows. Let $I = \{1, 2, \ldots, m\}$ be indices the rows, $J = \{1, 2, \ldots, n\}$ be indices the columns and

$$x_j = \begin{cases} 1 & \text{if column } j \text{ belongs to the solution} \\ 0 & \text{Otherwise} \end{cases} \quad \text{for } j \in J$$

$$\text{Minimize} \sum_{j \in J} c_j x_j \tag{1}$$

Subject to:

$$\sum_{j \in J} a_{ij} x_j \geqslant 1 \quad \text{for } i \in I \tag{2}$$

$$\text{and } x_j = 0 \text{ or } 1, \quad \text{for } j \in J \tag{3}$$

The following notations are often used for describing the set covering problem.

$J_i = \{j \in J : a_{ij} = 1\}$: the subset of columns covering row i.

$I_j = \{i \in I : a_{ij} = 1\}$: the subset of rows covered by column j.

The SCP is an NP-hard problem and many heuristic algorithms (Aickelin, 2002; Beasley & Chu, 1996; Caprara, Fischetti, & Toth, 1999; Chvatal, 1979; Lan, DePuy, & Whitehouse, 2006) have been developed to find a good solution for the SCP in a reasonable time. The genetic algorithm developed by Beasley and Chu (1996), the Lagaragian relaxation heuristic by Caprara et al. (1999) and Meta-RaPS heuristic by Lan et al. (2006) report the best solution quality.

This article is organized as follows. In Section 2 we discuss how randomness is introduced into basic Meta-RaPS along with the application for solving the Set Covering Problem. We present some new randomization methods in Section 3. In Section 4, we propose several memory mechanisms for Meta-RaPS, and then in Section 5 we compare the effectiveness of different randomness methods and memory mechanisms and analyze the interaction between these methods. In addition, our results are compared to other effective heuristics for the SCP. Finally, conclusions are drawn based on the experimental study and future research directions are presented in the last section.

2. Meta-RaPS – A template to introduce randomness

Meta-RaPS provides a template to introduce randomness systematically into multi-start heuristics such that properly diversified starting solutions with good solution quality can be generated and then the intensified neighborhood search, which is usually quite consuming, will only be applied to those promising starting solutions. The construction heuristic generates a feasible solution by adding basic elements step by step until a feasible solution is generated. The basic elements in a solution for a given combinatorial problem, such as the columns in a cover for the SCP, are described by a finite set $BE = \{1, 2, \ldots, n\}$ and the feasible basic elements ($FE \subseteq BE$) are those elements eligible for selection at each construction step. Each feasible basic element is characterized by a greedy score (P_i for $i \in FE$) according to some priority rule. Dependent on the definition of the priority rule, the best feasible element might assume either the lowest score or the highest score. Instead of always choosing the best feasible element as in greedy heuristics, Meta-RaPS introduces randomness via two parameters: %priority (percentage priority) and %restriction (percentage restriction). The parameter %priority determines the percentage of time that the best feasible element will be chosen. The remaining time, the element added to the solution will be randomly chosen from a candidate list (CL) that includes all elements considered 'acceptable'. The second parameter, %restriction, is used to determine the level of acceptance and thus the size of the candidate list. For example, suppose the best feasible element assumes the lowest greedy score for a particular priority rule, then at a specific iteration, if the element k is the best feasible element ($P_k = \min\{P_j : j \in FE\}$), then $CL = \{j : j \in FE \text{ and } P_j \leqslant P_k \times (1 + \%\text{restriction}/100)\}$. The smaller the %prior-

ity, the more randomness will be introduced. For a given %priority, the larger the %restriction, the more randomness will be introduced.

After a feasible solution is constructed, an improvement algorithm, usually a neighborhood search method, can be applied to improve the solution quality. The parameter %improvement (percent improvement) is used to define how many feasible solutions will be improved. Suppose Z^*_{con} is the objective function value of the best solution found before applying the improvement procedure, a constructed solution will be improved if its objective function value $Z \leqslant (1 + \%improvement) \times Z^*_{con}$ (for minimization problem). The underlying idea is the expectation that good unimproved solutions can lead to better neighboring solutions. The differences between Meta-RaPS and other similar algorithms, such as Greedy Algorithms, COMSOAL and GRASP have been investigated by DePuy, Moraga, and Whitehouse (2005). Using 100 %priority, and 0 %improvement, Meta-RaPS acts the same as Greedy Algorithms; Using 0 %priority, 0 %improvement and a large number of %restriction, Meta-RaPS mimics COMSOAL; Using Meta-RaPS with parameters of 0%priority and 100%improvement will imitate GRASP. So Meta-RaPS could be viewed as a general form of Greedy Algorithms, COMSOAL and GRASP.

To apply Meta-RaPS for solving the set-covering problem, we define the greedy score for each column j as: $P_j = c_j/k_j$ in the construction phase, where c_j is the cost of column j, and k_j is the number of currently uncovered rows that could be covered by column j, i.e. $k_j = |\{i : i \in I_j \setminus \cup_{n \in X} I_n\}|$. After a feasible solution is constructed, all redundant columns (Given X is a feasible solution and column $j \in X$, j is redundant if $X \setminus \{j\}$ is still feasible) will be removed from the solution such that the redundant column with largest cost will be removed at first. To improve the solution quality, a two-step neighborhood search procedure may be applied after construction. First, a number of columns are randomly removed from the given constructed solution. The number of removed columns is equal to $|X| \times$ %search_magnitude, where %search_magnitude is a user-defined parameter. Then the partial solution is made feasible by solving a reduced size SCP that is made up of the uncovered rows and the columns that could cover these rows. This reduced-size SCP is defined by

$$\min \left\{ \sum_{j \in J'} c_j x_j : \sum_{j \in J'} a_{ij} x_j \geqslant 1 (i \in I'), \quad x_j = 0 \text{ or } 1 (j \in J') \right\},$$

where

$$I' = I \setminus \bigcup_{m \in X} I_m \text{ and } J' = \bigcup_{i \in I'} J_i$$

The Meta-RaPS SCP construction heuristic is reused for solving this reduced-size SCP and a neighboring solution is obtained after combining the solution of this smaller SCP with the partial solution, and removing the redundant columns. We replace the original solution with this neighboring solution if it gives a lower cost, so the neighborhood search is always executed around the best solution found so far in the current iteration. This improvement algorithm is repeated a number of iterations (imp_iteration). The pseudo-code for the basic version of Meta-RaPS SCP is shown in Fig. 1.

3. More randomization strategies

In most multi-start heuristics, such as GRASP or the basic version of Meta-RaPS, we evaluate all feasible elements by using a single priority rule. The diversity and quality of solutions in the construction phase are controlled by adjusting some parameters such as %restriction and %priority. A new randomization method proposed here is to use multiple priority rules instead of a single priority rule to evaluate these feasible elements in the construction phase. More specifically, we present two strategies to apply multiple priority rules as follows.

- Multiple priority rules within an iteration (or intra-iteration randomization): before evaluating the feasible elements at each construction step in an iteration, a priority rule will be randomly selected from a pool of these priority rules, and this priority rule will be active only for the current construction step. So in this strategy, multiple priority rules will be used within an iteration.

```
procedure Meta-RaPS-SCP(I, J, %priority, %restriction, %improvement, max_iteration,
%search_magnitude, imp_iteration)
    z* = z*con = LARGE_NUMBER {set to a large number}
    for it = 1, …, max_iteration
      Construct a feasible solution based on the priority values: Pj = cj / kj
      if Z < z*con then z*con = Z
      if Z ≤ (1 + %improvement) × z*con then
          for imp = 1, …, imp_iteration
              Apply the two-step neighborhood search procedure to find better neighboring solutions
          end for
      if (Z < z*) then z* = Z
    end for
    return z*
end Meta-RaPS-SCP
```

Fig. 1. Pseudo-code of Meta-RaPS SCP algorithm.

- Multiple priority rules between iterations (or inter-iteration randomization): in this strategy, a single priority rule will be randomly chosen at the beginning of iteration and remain active in this iteration. However, different priority rules may be used in different iterations.

The idea here is to apply multiple priority rules to have a more comprehensive assessment for the basic elements. Because more than one priority rule is used, this method gives more probabilistic solutions. It is this increased randomness from the method that helps to search the regions neglected by using the single priority rule and lead to better solutions. It should be noted that we may apply different parameters for different priority rules since the best parameter settings for %priority and %restriction may be different for each priority rules. Furthermore, the probability distribution to choose the priority rules may not necessarily be uniform. Instead, a more reasonable probability distribution should be the one that favors more effective priority rules based on empirical results for each priority rule.

For the Set Covering Problem, we design multiple priority rules by varying the influence of c_j and k_j. In addition to apply the priority rule of c_j/k_j, the effect of changing the ratio of the influence of c_j and k_j is investigated. The following priority rules: c_j/k_j, c_j/k_j^2, $c_j^{1/2}/k_j$, and $c_j/k_j^{1/2}$ form a pool of priority rules. In the first priority rule, c_j and k_j have the same influence. In the second and third priority rules, k_j assumes a higher role than c_j, while in the last rule, c_j is attached more importance. Attaching more importance to k_j has two effects. First, those columns that can cover more rows will assume better priority values and be favored in the construction. Second, since k_j changes dynamically in each construction step, placing k_j a higher role can usually increase the variance of generated solutions and thus may also be viewed as a randomization method. Since for the Set Covering Problem applying the same parameters for all these priority rules and applying the uniform distribution in choosing these priority rules can generate very good solutions (Lan et al., 2006), we have not selected the more complicated and consuming methods for setting parameters and assigning priority rules.

4. Incorporating adaptive memory mechanisms

In the basic Meta-RaPS, iterations are completely independent of each other. One way to enhance Meta-RaPS is to incorporate memory mechanisms in the construction phase which learn from previously generated solutions. The historical information about the basic elements is useful for the construction of new solutions. An information measure, element fitness, is used to record the historical information to be incorporated into Meta-RaPS construction phase. Notice that while we usually use the fitness to describe the quality or frequency of the previously generated solutions in evolutionary heuristics such as the Genetic Algorithm, the fitness proposed here is calculated for basic elements of the solutions. The element fitness is designed based on the ideas of Tabu Search (Glover & Laguna, 1997). Suppose S is a list of elite solutions that have good solution

quality and proper diversity, and there are r solutions in S. Let $S_j \subset S$ denote the set of elite solutions that include the basic element j. The fitness can be defined as follows.

$$e_j = \left\{\sum_{s\in S_j} 1/Cost(s)\right\} \Big/ \left\{\sum_{s\in S} 1/Cost(s)\right\},$$

where $Cost(s)$ is the objective function value of solution s.

The more frequently the column j is included in the elite solutions, and the better the solutions are, the larger the value of e_j. The value of e_j is always less than or equal to 1. The list of elite solutions S is updated dynamically; when a solution is better than the worst of the elite solutions, it will be a candidate for S. It replaces the worst elite solution if it is sufficiently different from these elite solutions. For example, the solution to be added must be different from any solutions in the elite list by at least a certain number of columns, called level of diversity (d). Otherwise, the solution is discarded unless an aspiration criterion is satisfied, i.e. the solution is better than any one of the elite solutions. So, the fitness of basic elements contains the quality information of the associated solutions, frequency of the selection, and is adjusted by its contribution to the diversity of the searching space. The size for the elite solution set (r) and the level of diversity (d) can be determined by the trial-and-error method.

Based on the fitness of each basic element, two methods to introduce memory into Meta-RaPS are designed. One is to incorporate the element fitness into priority rules and the other is called partial construction in which the values of some variables are completely determined by element fitness.

4.1. Priority rules with element fitness

The first memory method is to design the priority rule as a function of the element fitness. For the Set Covering Problem, one such priority rule is $c_j/\{(1 + le_j)k_j\}$, where l is a parameter to control the influence of the element fitness. As more iterations are run for the algorithm, more information about each basic element is gathered. In the earlier stage, the memory information is not complete and a smaller factor of l is used. Since incorporating fitness emphasizes solution quality at the expense of randomness, it is coupled with some diversification components that periodically encourage the search to explore different regions of the search space. In this research, diversification can be achieved by periodically setting l to zero to increase the diversity of generated solutions.

4.2. Partial construction

In the method of partial construction, some basic elements are fixed from previous iterations. The degree of partial construction (%partial) indicates the number of elements that remain fixed from one iteration to another. Since more information is gathered to evaluate the remaining elements after fixing those elite basic elements, more correct decisions are expected regarding the selection of these remaining elements. Partial construction can also be considered as an instance of the path relinking method (Glover & Laguna, 2000) that constructs new solutions by exploring trajectories that connect high-quality solutions. When applying this method to the SCP, those basic elements with a positive fitness are considered elite elements and the set of elite elements are updated with the updating of the list of elite solutions. Since at the beginning of the algorithm, the memory information is not complete and maybe misleading, we run a certain number of iterations without applying memory (trial-period). After this trial period, the new solutions are constructed by first randomly selecting a number of elements from the elite set, and then completing the solution by the usual construction method. The elite elements can be selected according to some probability distribution to form the partial solution. We apply the uniform distribution to select the elite elements for its simplicity and effectiveness in this study. In comparison with the method to introducing memory into the priority rules, this partial construction method is simpler as it does not include any additional fitness calculations. Since only a certain percentage of columns (100-%partial) need to be chosen based on the priority values, the construction heuristic works much faster.

5. Computational results and analysis

In this section, we report numerical results performed on the 65 SCP instances in the Beasley's OR Library (1990). The experiments are conducted on different variants of Meta-RaPS, namely, basic Meta-RaPS, Meta-

RaPS with adaptive memory (mMeta-RaPS) as described in Sections 4.1 and 4.2, and Meta-RaPS with new randomization strategies (rMeta-RaPS) presented in Section 3, as well as the Meta-RaPS with both memory mechanisms and new randomization methods (mrMeta-RaPS). We report the results for both Meta-RaPS construction (without improvement) and Meta-RaPS with improvement, because it is discovered that the performance of these methods appears somewhat different under the two situations. Finally, the results of these algorithms are compared to several other effective heuristics from the literature.

5.1. Algorithms

We classified different variants of Meta-RaPS into four classes: Meta-RaPS, Meta-RaPS with memory (mMeta-RaPS), Meta-RaPS with new randomization strategies (rMeta-RaPS), and Meta-RaPS with memory and randomization (mrMeta-RaPS). Table 1 lists different algorithms in these classes with each one assigned a code. We use four different priority rules in the basic Meta-RaPS. The priority rules c_j/k_j and c_j/k_j^2 are chosen to apply the memory mechanisms because these two priority rules work the best for the Meta-RaPS without improvement (construction) and Meta-RaPS with improvement, respectively. The two memory mechanisms, priority rule with fitness and partial construction are tested, so there are four algorithms in mMeta-RaPS. The interaction effect of the randomization methods and the partial construction method, which is a more effective memory mechanism than priority rules with fitness, is investigated through experiments on mrMeta-RaPS.

5.2. Test problems and parameter settings

The SCP instances are divided into seven sets and each set contains five or 10 problems. The sizes of these problems range from 200 (rows) × 1000 (columns) to 400 (rows) × 4000 (columns) as described in Table 2. In

Table 1
Algorithms tested

Classification	Alg. #	Priority rule	Memory type
Meta-RaPS	1	Single (c_j/k_j)	None
	2	Single (c_j/k_j^2)	None
	3	Single ($c_j^{1/2}/k_j$)	None
	4	Single ($c_j/k_j^{1/2}$)	None
mMeta-RaPS	5	Single (c_j/k_j)	Fitness priority rule: $c_j/(1+le_j)k_j$
	6	Single (c_j/k_j)	Partial construction
	7	Single (c_j/k_j^2)	Fitness priority rule ($c_j/(1+le_j)k_j^2$)
	8	Single (c_j/k_j^2)	Partial construction
rMeta-RaPS	9	Intra-iteration randomization	None
	10	Inter-iteration randomization	None
mrMeta-RaPS	11	Intra-iteration randomization	Partial construction
	12	Inter-iteration randomization	Partial construction

Table 2
Test instances

Set Name	No. of instances	No. of rows (m)	No. of columns (n)	Range of cost	Density
4	10	200	1000	1–100	2%
5	10	200	2000	1–100	2%
6	5	200	1000	1–100	5%
A	5	300	3000	1–100	2%
B	5	300	3000	1–100	5%
C	5	400	4000	1–100	2%
D	5	400	4000	1–100	5%
NRE	5	500	5000	1–100	10%
NRF	5	500	5000	1–100	20%
NRG	5	1000	10,000	1–100	2%
NRH	5	1000	10,000	1–100	5%

Table 3
Parameter settings

Meta-RaPS	Priority rules with fitness	Partial construction
%Priority and %restriction: 20 combinations	Size of the elite solution: $r = 5$	
%Improvement = 15	Level of diversity: $d = 1$	Trial-period = 20
%Search_magnitude = 30	Fitness factor:	%Partial = 70
Iteration: 100		
Imp_iteration: 400	$l = \min\{1.0,\ 0.1 \times (i/20) \times I(i\%20 = 0)\}$	

Table 2 'Density' is the percentage of non-zero entries in the SCP matrix and 'Range of cost' shows the minimum and maximum cost of the columns. Note the sizes of the Set E, F, G, and H are very large, therefore we chose not to compare the performance of different variants of Meta-RaPS on these sets. However, when comparing Meta-RaPS with other heuristics, we reported the results for all problem sets for the best Meta-RaPS variant, namely, mrMeta-RaPS.

The parameter settings are listed in Table 3. Two key parameters for the construction phase are %priority and %restriction. To block the factors of %priority and %restriction when comparing different algorithms, we apply different parameter settings for %priority (5, 10, 15, 20) and %restriction (5, 15, 25, 35, 45) and then take the average over these parameter settings. The best parameter settings from these 20 combinations are selected when comparing with other heuristics in the literature. Coy, Golden, Runger, and Wasil (2001) note that it is often difficult to find appropriate parameter settings for meta-heuristics and common procedures have ranged from simple trial and error to complicated sensitivity analysis. A trial and error procedure was used in this research to set the parameters. The first instances in problem sets 4, 5, 6, A, B, C and D are selected as representative and different combinations of parameters are applied for these instances. More specifically, we take %improvement from {10, 15, 20} and imp-magnitude from {0.3, 0.4}. The parameter max_iteration is chosen from {100, 500, 1000} and imp_iteration is chosen from {300, 400, 500}. For priority rules with fitness, the size for the list of elite solutions is chosen from {2, 5, 10}. The level of diversity is chosen from {1, 5, 10}. For the parameter l, in the earlier stage, the memory information is not complete and a smaller factor of l is used. For example, the value of l is increased 0.1 every 20 iterations to apply the increasing influence of the memory information. l is fixed at 1.0 after 200 iterations, so l can be represented by $\min\{1.0,\ 0.1 \times (i/20) \times I(i\%20 = 0)\}$, where i is the iteration, and $I(i\%20 = 0)$ is the identity function. We also apply $l = \min\{2.0,\ 0.2 \times (i/20) \times I(i\%20 = 0)\}$ to increase the influence of the memory in the trial-and-error method. For the partial construction, we choose the trial-period from {10, 20, 30} and choose %partial from {50, 70, 80}. The parameter setting obtained from this trial-and-error method is listed in Table 3.

5.3. Results and analysis

We first report the summarized results for all the 45 instances in sets 4, 5, 6, A, B, C, D to have an overall comparison of these algorithms. In order to illustrate the effect of various randomization methods and memory mechanisms graphically, the results are also presented for a single SCP instance (Problem A.3) in Fig. 2. Tables 4 and 5 are the summarized test results for the Meta-RaPS construction and the Meta-RaPS with improvement, respectively. In these tables, gap is the percentage of the deviation from the optimal solution. Solution time is the time to find the best solution for the first time. The Avg. Mean, Avg. Std. Dev., Avg. gap and Avg. Sol. time (in Intel P4 1.7 GHz CPU seconds) are the average values taken over all test instances and the 20 different parameter settings for %priority and %restriction. The Avg. No. Optima is the average number of optimal solutions found using these parameter settings. The Max. No. Optima is the maximum number of optimal solutions generated by the best parameter setting.

One obvious observation from these test results is the significant increase in solution quality brought forth by the improvement heuristic. The improvement on the Avg. gap ranges from 58% to 99%. Several more explicit and interesting findings are listed as follows.

First, for the four basic Meta-RaPS algorithms with different priority rules (Algorithms 1–4 in Table 4), c_j/k_j^2 and $c_j^{1/2}/k_j$ are not as effective as c_j/k_j and $c_j/k_j^{1/2}$ for Meta-RaPS construction. However, after the

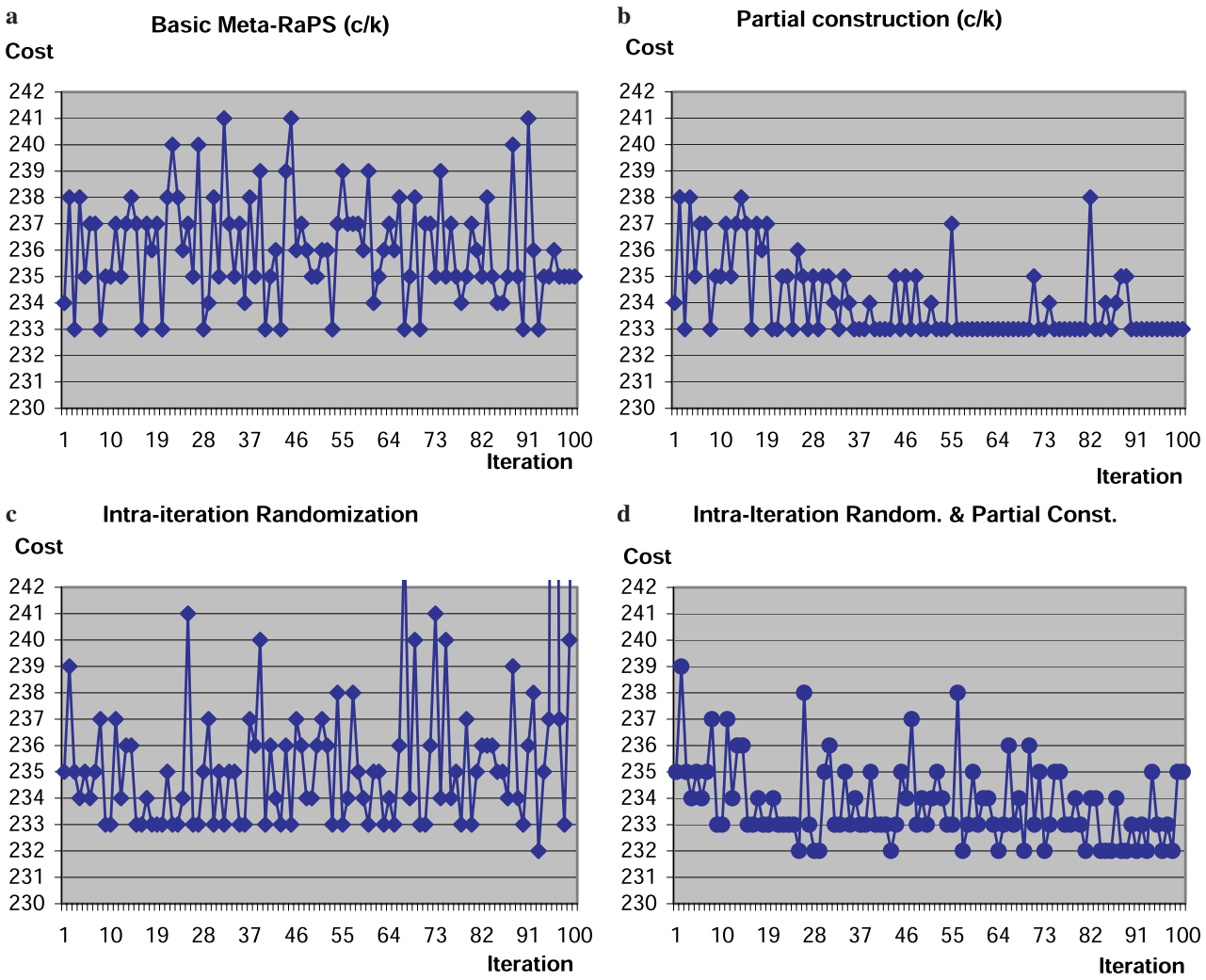

Fig. 2. Iterative results for Problem A.3.

Table 4
Results for Meta-RaPS SCP without improvement

Alg. #	1	2	3	4	5	6	7	8	9	10	11	12
Avg. Mean	271	286	293	275	264	268	275	282	282	281	282	274
Avg. Std. Dev.	5.28	9.93	12.44	5.47	5.20	6.36	9.53	11.82	10.52	13.36	10.51	10.61
Avg. gap	2.6	5.19	5.92	3.96	2.1	1.86	4.11	3.63	3.36	2.9	2.65	2.16
Avg. Sol. Time	0.09	0.12	0.16	0.18	0.10	0.10	0.12	0.13	0.11	0.10	0.11	0.10
Avg. No. Optima	1.75	0.8	0.45	1.35	3.65	4.15	1.95	2.15	1.5	1.6	3.85	3.9
Max No. Optima	4	3	2	2	6	8	3	4	4	3	10	9

Table 5
Results for Meta-RaPS SCP with improvement

Alg. #	1	2	3	4	5	6	7	8	9	10	11	12
Avg. Mean	256	260	265	260	256	256	256	256	259	263	256	257
Avg. Std. Dev.	1.66	10.32	17.51	1.90	0.97	1.69	5.0	5.44	10.36	15.06	4.84	7.21
Avg. gap	0.294	0.037	0.037	1.649	0.359	0.28	0.27	0.026	0.008	0.018	0.015	0.010
Avg. Sol. Time	2.57	2.02	2.24	3.34	1.63	2.83	1.66	2.12	2.24	2.67	1.99	2.49
Avg. No. Optima	31.35	41.35	41.65	12.8	28.45	31.7	40	42.35	44.25	43.25	43.55	43.9
Max. No. Optima	40	44	44	25	38	40	42	45	45	45	45	45

improvement heuristic is applied, these two priority rules become more effective in terms of average gap and average solution time (see Table 5).

Second, introducing memory into Meta-RaPS (Algorithms 5–8) can improve the solution quality for the construction phase. For example, for the basic Meta-RaPS algorithm c_j/k_j (Algorithm 1), the average gap is decreased 19.23% if incorporating fitness into the priority rule (Algorithm 5), and decreased 28.46% if the partial construction (Algorithm 6) is applied. Clearly, the partial construction method is more effective than introducing fitness into priority rules. However, if the improvement heuristic is applied, only the algorithm c_j/k_j^2 with partial construction (Algorithm 8) improves the solution quality over the corresponding basic Meta-RaPS algorithm c_j/k_j^2 (Algorithm 2).

Third, for the construction heuristic, none of the randomization methods (Algorithms 9 and 10) can perform better than the basic Meta-RaPS with priority rule c_j/k_j in terms of solution quality. However, these new randomization methods do greatly better the solution quality for Meta-RaPS with improvement. For example, the Avg. gap in algorithm 9 (Intra-iteration randomization) decreases 78.38% over that in Algorithm 2 (c_j/k_j^2), which is the best among the basic Meta-RaPS algorithms if the improvement heuristic is applied.

Fourth, when we apply the randomization and partial construction methods at the same time (Algorithms 11 and 12), the effectiveness of both Meta-RaPS construction and Meta-RaPS with improvement can be bettered. For the Meta-RaPS construction phase, although the algorithms 11 and 12 cannot significantly improve the Avg. gap, they provide the maximal number of optimal solutions among all these construction algorithms. For the Meta-RaPS with improvement, incorporating the partial construction into randomization methods can decrease the average solution time while maintaining good performance on the solution quality. For example, the average solution time is decreased from 2.24 s in algorithm 9 to 1.99 s in algorithm 11.

Fig. 2 gives the iterative results for the Problem A.3 that has an optimal cost of 232. We show how the randomization and memory strategies work by comparing the results from four different algorithms; Basic Meta-RaPS (Algorithm 1), Meta-RaPS with partial construction (Algorithm 6), Meta-RaPS with intra-iteration randomization (Algorithm 9) and Meta-RaPS with intra-iteration randomization and partial construction (Algorithm 11). The parameters are: %priority = 5 and %restriction = 35, and all other parameters are the same as in Section 5.1. In the basic Meta-RaPS (Fig. 2a), the results from different iterations are independent from each other, and the best solution has a cost of 233. After the partial construction is introduced between iterations (Fig. 2b), the variance of generated solution is decreased and the algorithm converges to the local optimal solution. On the other hand, when we introduce intra-iteration randomization into basic Meta-RaPS (Fig. 2c), the variance of generated solutions is increased and the optimal cost of 232 is found in the 93rd iteration. With the collaboration of the partial construction and intra-iteration randomization (Fig. 2d), the optimal cost is found in the 25th iteration, and we found the optimal solution quite frequently after that, so the solution time has been greatly decreased. However, it should be noted that if Meta-RaPS with intra-iteration randomization can find the optimal solution during the trial period (20 iterations), incorporating the partial construction would not shorten the time to find the best solution. That is the reason why the decrease in average solution time for Algorithms 11 and 12 over Algorithms 9 and 10 in Table 5 is not so great as might be expected from the comparison of Fig. 2c and d.

In summary, for the Meta-RaPS construction, the priority rules with less randomness tend to generate better results and introducing memory can significantly improve the solution quality. For Meta-RaPS with improvement, the priority rules with more randomness always dominate those with less randomness. In this situation, introducing memory does not necessarily improve the solution quality. However, the use of memory improves the efficiency by shortening the time to find the best solution if the intra-iteration or inter-randomization is applied. Memory also decreases the mean and variance of the solution values. Actually, the most successful algorithms are those with a good balance between randomness and memory. As for the SCP, if the improvement heuristic is applied, using Algorithms 8, 9, 10, 11, 12 can find all the optimal solutions for the test instances.

5.4. Comparison with other heuristics from the literature

Meta-RaPS with both intra-iteration randomization and partial construction (Algorithm 11, with %priority = 5 and %restriction = 45) is compared to the following effective SCP heuristics from the literature.

Table 6
Solution quality of different algorithms

	Meta-RaPS	CFT	BeCh	IGA	GRASP	Be	PROGRES
No. Opt.	65/65	65/65	61/65	61/65	4065	20/65	0/65
Avg. gap	0	0	0.013	0.046	0.35	0.537	1.106

- Be: the Lagrangian heuristic by Beasley (1990)
- BeCh: the genetic algorithm by Beasley and Chu (1996)
- CFT: the Lagrangian heuristic by Caprara et al. (1999)
- PROGRES: a probabilistic greedy search heuristic by Haouari and Chaouachi (2002)
- IGA: an indirect genetic algorithm by Aickelin (2002)
- Reactive GRASP with path relinking (GRASP)

To compare Meta-RaPS with GRASP, we implement a reactive GRASP with path-relinking. The basic GRASP procedure is simulated by Meta-RaPS by setting %priority = 0 and %improvement = 100. For reactive GRASP, the parameter %restriction is randomly chosen from the set R = {5, 15, 25, 35, 45}. Initially these parameters are chosen with equal probability, and then for every 20 iterations these probabilities are updated as follows.

$$P(\%restriction = R_i) = \frac{1/C(R_i)}{\sum_1^5 (1/C(R_i))} \quad \text{for } 1 \leqslant i \leqslant 5,$$

where $C(R_i)$ is the average cost for the previously generated solutions using %restriction = R_i.

The path relinking we used is the standard forward linking method (Resende & Ribeiro, 2005) in which the two-step neighborhood search is applied successively from X_1 to X_2, where X_1 is the starting solution and X_2 acts as the guiding solution from the elite list. In this two-step neighborhood search, we add the columns $j \in X_2 \backslash X_1$ as the feasible elements successively and a certain number of iterations (30) for neighborhood search are executed once a new feasible element is added.

Table 6 compares the solution quality of Meta-RaPS with these heuristics. The number of optimal solutions found by these heuristic and the average gap are reported. Meta-RaPS SCP found all the optimal the 65 non-unicost problems in OR-library. It is one of only two heuristics in the literature that found all the optimal solutions.

A comparison of computation times for different heuristics is difficult since the computational efficiency is affected by many factors such as the implementation language, the complier, the executing environment including CPU, RAM, operating system, as well as the programmer's skills. For an approximate comparison, the actual computation times from different computers is often 'converted' to that on a benchmark computer. For example, Caprara et al. (1999) adjusted the computation time of many algorithms and heuristics for SCP to DECstation 5000/240 s according to the performance report by Dongarra (2004). From our experiments on an Intel PIV 1.7 GHz PC and an Intel PII 400 MHz PC, the average error associated with using this Dongarra (2004) conversion could be as high as 52.43%. Therefore, we report both the actual computation time in different computers and the transformed time to DECstation 5000/240 s. Notice that the computation time for BeCh is calculated according to the method proposed by Caprara et al. (1999) since the computation time has only been reported for different runs for the algorithm. From this table, it can be concluded that the mrMeta-RaPS algorithm is slower than CFT, but much faster than BeCh (notice that BeCh has not found the best known solutions for Set G and Set H, although it outperforms in computation time for Set H). In comparison with CFT, we stress that the main advantage of the Meta-RaPS SCP algorithm for practitioners is its simplicity. This algorithm consists of introducing random factors in a greedy algorithm, together with a local search procedure to improve the solution found by the greedy algorithm. In this implementation we also use other techniques such as simple randomization and memory methods. We also show, through the use of benchmark SCP test problems, this simplistic algorithm is comparable in terms of solution quality to complex, state-of-the-art SCP solution techniques (Table 7).

Table 7
Comparison of computation time

Problem set	CFT Actual time on DECstation 5000/240 (s)	mrMeta-RaPS		BeCh	
		Actual time on Intel PIV 1.7 G (s)	Expected time on DECstation 5000/240 (s)	Actual time on Graphics Indigo (R4000,100) (s)	Expected time on DECstation 5000/240 (s)
4	6.5	0.315	21.57	57.59	163
5	3.2	0.188	12.872	190.87	540.2
6	9.4	0.122	8.368	20.21	57.2
A	106.6	2.018	138.21	52.79	149.4
B	7.4	1.764	120.82	54.9	155.4
C	66	3.844	263.27	70.38	199.2
D	17.2	4.264	292.04	81.41	230.4
E	118.2	8.46	579.42	3082.55	8724.2
F	109	53.47	3662.16	976.90	2764.8
G	504.8	175.64	12029.58	4540.83	12851.4
H	858.2	543.79	37244.17	2240.70	6341.6
Overall	164	72.17	4249.95	1033.5	2925

6. Conclusions and future research

This paper presents and compares different variants of a multi-start metaheuristic, Meta-RaPS, with a focus on analyzing the effects of randomness and memory. It is found that for the memory mechanisms, the method of partial construction works better than priority rule with element fitness. Incorporating memory into Meta-RaPS can improve the solution quality for the construction phase. However, it does not necessarily improve the solution quality of the overall algorithm for Meta-RaPS after the intensified search of the improvement phase is applied. On the contrary, the two methods for introducing more randomness into Meta-RaPS, intra-iteration and inter-iteration randomization, does not significantly improve the solution quality for the construction phase. However, it greatly improves the solution quality for the overall algorithm after the improvement heuristic is applied. By incorporating the partial construction method into Meta-RaPS with intra-iteration or inter-iteration randomization, the solution quality for the construction phase can be increased in some sense and the computation time to find the best solution in the overall algorithm can be decreased. All these findings show the importance of maintaining a good balance between the diversification and intensification in multi-start heuristics. This paper proposes several methods for achieving a good balance for introducing randomness and memory into Meta-RaPS construction and several effective algorithms developed in this paper can find all the optimal solutions for the SCP test instances. Such strategies for introducing randomness and memory can be applied to other application problems for Meta-RaPS and even other multi-start local search heuristics. Future research should also be directed toward better integration between randomization methods and memory mechanisms. For example, the issue of how to improve the utility of the memory information in the trial period of partial construction needs to be addressed.

References

Aickelin, U. (2002). An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society, 53*(10), 1118–1126.

Arcus, A. L. (1966). COMSOAL: a computer method of sequencing operations for assembly lines. *International Journal of Production Research, 4*(4), 259–277.

Beasley, J. E. (1990). A Lagrangian heuristic for set covering problems. *Naval Research Logistics, 37*(1), 151–164.

Beasley, J. E., & Chu, P. C. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research, 94*, 392–404.

Boese, K. D., Kahng, A. B., & Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters, 16*(2), 101–113.

Caprara, A., Fischetti, M., & Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research, 47*(5), 730–743.

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of Operations Research, 4*(3), 233–235.

Coy, S., Golden, B., Runger, G., & Wasil, E. (2001). Using Experimental Design to Find Effective Parameter Setting for Heuristics. *Journal of Heuristics, 7*, 77–97.

DePuy, G. W., Whitehouse, G. E. & Moraga, R. J. (2002). Using the Meta-Raps Approach to Solve Combinatorial Problems, *Proceedings of the 2002 Industrial Engineering Research Conference*, Orlando, Florida.

DePuy, G. W., Moraga, R. J., & Whitehouse, G. E. (2005). Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E: Logistics and Transportation Review, 41*(2), 115–130.

Dongarra, J. J. (2004). Performance of various computers using standard linear equations software. Technical Report No. CS-89-85, Computer Science Department, University of Tennesse.

Feo, T., Mauricio, G., & Resende, C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6*, 109–133.

Fleurent, C., & Glover, F. (1999). Improved constructive multi-start strategies for the Quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing, 11*, 198–204.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic Publishers.

Glover, F., & Laguna, M. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics, 29*(3), 653–684.

Haouari, M., & Chaouachi, J. S. (2002). A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem. *Journal of the Operational Research Society, 53*(7), 792–799.

Lan, G., DePuy, G. W., & Whitehouse, G. E. (2006). An Effective and Simple Metaheuristic Heuristic for the Set Covering Problem. *European Journal of Operational Research*, in press.

Martí, R. (2003). Multi-start methods. In Glover & Kochenberger (Eds.), *Handbook of MetaHeuristics* (pp. 355–368). Boston: Kluwer Academic Publishers.

Patterson, R., Rolland, E., & Pirkul, H. (1999). A memory adaptive reasoning technique for solving the capacitated minimum spanning tree problem. *Journal of Heuristic, 5*, 159–180.

Pitsoulis, L. S., & Resende, M. G. C. (2002). Greedy randomized adaptive search procedures. In P. M. Pardalos & M. G. C. Resende (Eds.), *Handbook of applied optimization* (pp. 168–183). Oxford: Oxford University Press.

Resende, M. G. C., & Ribeiro, C. C. (2003). Greedy randomized adaptive search procedures. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 219–249). Boston: Kluwer Academic Publishers.

Resende, M. G. C., & Ribeiro, C. C. (2005). GRASP with path-relinking: recent advances and applications. In T. Ibaraki, K. Nonobe, & M. Yagiura (Eds.), *Metaheuristics: Progress as Real Problem Solvers* (pp. 29–63). Berlin: Springer.