A Variable Depth Search Algorithm for the Generalized Assignment Problem

Mutsunori Yagiura[1], Takashi Yamaguchi[1] and Toshihide Ibaraki[1]

[1]Department of Applied Mathematics and Physics
Graduate School of Engineering, Kyoto University
Sakyo-ku, Kyoto 606-01, Japan
Emails: {yagiura, ibaraki}@kuamp.kyoto-u.ac.jp

Abstract: A *variable depth search* procedure (abbreviated as VDS) is a generalization of the local search method, which was first successfully applied by Lin and Kernighan to the traveling salesman problem and the graph partitioning problem. The main idea is to adaptively change the size of neighborhood so that it can effectively traverse larger search space while keeping the amount of computational time reasonable. In this paper, we propose a heuristic algorithm based on VDS for the generalized assignment problem, which is one of the representative combinatorial optimization problems known to be NP-hard. To the authors' knowledge, most of the previously proposed algorithms (with some exceptions) conduct the search within the feasible region; however, there are instances for which the search within feasible region is not advantageous because the feasible region is very small or is combinatorially complicated to search. Therefore, we allow in our algorithm to search into the infeasible region as well. We also incorporate an adaptive use of two different neighborhoods, shift and swap, within the sequence of neighborhood moves in VDS. Computational experiments exhibit good prospects of the proposed algorithm.

Keywords: adaptive neighborhood, generalized assignment problem, local search, metaheuristics, variable depth search procedure

1 Introduction

To deal with computationally hard problems, approximate algorithms are used to provide reasonably good solutions in practical time. A *variable depth search* procedure (abbreviated as VDS) is a generalization of the local search method. The VDS was first successfully applied by Lin and Kernighan to the traveling salesman problem and the graph partitioning problem [4, 6]. The main idea is to adaptively change the size of neighborhood so that the algorithm can effectively traverse larger search space while keeping the amount of computational time reasonable. In VDS, the neighborhood is usually defined to be the set of solutions obtained by a sequence of $\bar{d}$ simple neighborhood moves, where $\bar{d}$ is a parameter adaptively changed in the algorithm. It is important to include problem specific characteristics into the VDS to make the adaptive change successful.

In this paper, we propose a heuristic algorithm based on VDS for the generalized assignment problem (abbreviated as GAP), which is one of the representative combinatorial optimization problems known to be NP-hard. In GAP, given n jobs and m agents, we are asked to determine a minimum cost assignment such that every job is assigned to exactly one agent and the resource constraint for each agent is satisfied. Among various heuristic algorithms developed for GAP are: a combination of the greedy method and the local search by Martello and Toth [7, 8]; a tabu search

and simulated annealing by Osman [10]; a genetic algorithm by Chu and Beasley [2]; a VDS by Racer and Amini [11]; a tabu search based on ejection chain approach by Laguna et al. [5] (which is proposed for a generalization of GAP); and so on.

To the authors' knowledge, many of the previous algorithms conduct the search only within the feasible region. However, we often encounter instances for which the search within feasible region is difficult because the feasible region is very small or is combinatorially complicated to search. In fact, just finding a feasible solution for GAP is already known to be NP-hard. Therefore, we allow in our algorithm to search into the infeasible region as well. It should be noted that the tabu search by Laguna et al. [5], in which strategic oscillation approach is employed, also allows the search into the infeasible region. It is also noted that the VDS by Racer and Amini [11] also allow the search into the infeasible region by introducing a dummy agent with large cost; but in their case the search never returns to the infeasible region once a feasible solution is found.

We incorporate an adaptive use of two different neighborhoods, shift and swap. The shift neighborhood is defined to be the set of solutions obtainable by changing the assignment of one job, while the swap neighborhood is defined to be the set of solutions obtainable by exchanging the assignments of two jobs. We propose to use two neighborhoods alternately and adaptively within the sequence of neighborhood moves in VDS.

Computational experiments were conducted on benchmark problem instances with up to $n = 200$. The results indicate that the proposed algorithm produces solutions of comparable or better quality in reasonable amount of time for many of the tested problem instances, compared to the existing heuristic algorithms as cited above.

2 Generalized Assignment Problem

Given n jobs $J = \{1, \ldots, n\}$ and m agents $I = \{1, \ldots, m\}$, we are asked to determine a minimum cost assignment such that every job is assigned to exactly one agent and the resource constraint for each agent is satisfied. We are also given the cost c_{ij} and the resource requirement a_{ij} if job j is assigned to agent i, and the amount b_i of the resource available to agent i. An assignment is a mapping σ: $J \rightarrow I$, where $\sigma(j) = i$ means that job j is assigned to agent i. The problem is formulated as follows:

$$\begin{aligned} \text{minimize} \quad & cost(\sigma) := \sum_{j \in J} c_{\sigma(j),j} \\ \text{subject to} \quad & \sum_{j \in J, \sigma(j)=i} a_{ij} \leq b_i, \forall i \in I. \end{aligned}$$

GAP is known to be NP-hard (e.g., [12]). It can be shown that just finding a feasible solution for GAP is NP-hard, since the partition problem [3] can be reduced to this problem with $m = 2$.

3 Local Search

VDS is a generalization of local search. The local search starts from an initial solution σ and repeats replacing σ with a better solution in its *neighborhood* $N(\sigma)$ until no better solution is found in $N(\sigma)$, where $N(\sigma)$ is a set of solutions obtained

from σ by slight perturbations. A solution σ is called *locally optimal*, if no better solution exists in $N(\sigma)$. The local search is often applied to a number of randomly generated initial solutions, and the best among the obtained local optimal solutions is output. This is called the *random multi-start local search*.

We use two types of neighborhoods, shift neighborhood N_{shift} and swap neighborhood N_{swap}, in our algorithm, where

$N_{shift}(\sigma) := \{\sigma' |\ \sigma'$ is obtained from σ by changing the assignment of one job$\}$,
$N_{swap}(\sigma) := \{\sigma' |\ \sigma'$ is obtained from σ by exchanging the assignments of two jobs$\}$.

As n is usually much larger than m, the size of shift neighborhood $|N_{shift}| = O(mn)$ is considered to be much smaller than that of swap $|N_{swap}| = O(n^2)$. In a shift move applied to an assignment σ, the resource requirement for the agent to which a job is added may increase greatly; while, in a swap move, resource requirements of the two relevant agents may not change much if the requirements of the two jobs are close. Note that, since the number of jobs assigned to each agent will not change if we use swap moves only, we should combine swap moves with shift moves in order to search different types of assignments. Based on these observations, we propose to use two neighborhoods alternately and adaptively within the sequence of neighborhood moves in VDS.

It is often effective to allow the search to visit infeasible region as well, in particular if the problems have tight constraints. In this case, it may be a good idea to evaluate solutions by the objective function penalized by infeasibility. Here we use the following function:

$$pcost(\sigma) := cost(\sigma) + \alpha \sum_{i \in I} p_i(\sigma), \tag{1.1}$$

where

$$p_i(\sigma) := \max \left\{ 0, \left(\sum_{j \in J, \sigma(j)=i} a_{ij} \right) - b_i \right\}$$

and α (> 0) is a prespecified parameter. The locally optimal solution under this function may not always be feasible; however, we can raise the probability of obtaining feasible solutions by (1) keeping the best feasible solution obtained during the search as an incumbent solution, and (2) using sufficiently large α.

The local search, in which N is used for the neighborhood (e.g., $N = N_{shift}$ or N_{swap}), f is used to evaluate solutions (e.g., $f = cost$ or $pcost$), and an initial solution σ (not necessarily feasible) is given, is formally described as follows.

Algorithm $LS(N, f, \sigma)$

Step 1 If there is a solution $\sigma' \in N(\sigma)$ such that $f(\sigma') < f(\sigma)$, set $\sigma := \sigma'$ and return to Step 1. Otherwise go to Step 2.

Step 2 ($f(\sigma') \geq f(\sigma)$ holds for all $\sigma' \in N(\sigma)$.) Output σ and stop.

We implement Step 1 by the first admissible move strategy, in which solutions in $N(\sigma)$ are scanned according to a prespecified random order and the first improved solution is immediately accepted as the next solution.

4 Variable Depth Search

An important feature of our VDS is the alternate use of shift and swap neighborhoods, which will be explained later in this section. To understand this, consider first the behavior of local search in which $N_{shift} \cup N_{swap}$ is always used as the neighborhood. In this case, if the current solution σ is close to locally optimal, shift moves rarely occur, since the agent i to which a job is shifted usually becomes infeasible and its penalty $p_i(\sigma)$ increases a large amount. This means that, as swap moves do not change the number of jobs assigned to each agent, the number of jobs of each agent is usually fixed at an early stage of the search and will never be changed until a locally optimal solution is reached. In other words, local search with neighborhood $N_{shift} \cup N_{swap}$ may not have enough search power even if the size $|N_{shift} \cup N_{swap}|$ is fairly large.

In order to vest diversifying power to our VDS, we systematically change the number of jobs assigned to each agent by employing the alternating use of shift and swap neighborhoods. Our algorithm forces to execute shift moves (which are usually degrading moves) as soon as a locally optimal solution is obtained in the sense of swap neighborhood, and these two phases of local search are repeated until some stopping criterion is satisfied.

In VDS, an enlarged neighborhood is used, which is usually defined to be the set of solutions obtained by a sequence of $\bar{d}$ simple neighborhood moves, where $\bar{d}$ is a parameter adaptively changed in the algorithm. Our VDS starts from a solution $\sigma^{(0)} = \sigma_0$ which is already locally optimal with respect to N_{swap}. The first move from σ_0 to σ_1 is therefore a shift, followed by swap moves $\sigma_1 \rightarrow \sigma_2$, $\sigma_2 \rightarrow \sigma_3$, ...to a locally optimal solution $\sigma^{(1)} = \sigma_k$. This loop of a shift move and swap moves is iterated $\bar{d}$ times to generate a sequence of solutions $\sigma^{(0)}, \sigma^{(1)}, \ldots, \sigma^{(\bar{d})}$. Then the best one among $\sigma^{(0)}, \sigma^{(1)}, \ldots, \sigma^{(\bar{d})}$ is chosen as the initial solution for the next sequence of solutions. These sequences are organized into three nested loops in our VDS, as described below. Some part of the algorithm, such as the modified shift and swap neighborhoods, $N'_{shift}(\sigma)$ and $N^{dsj}_{swap}(\sigma_k, I_k)$, and the set I_k, will be explained later in more details.

Algorithm VDS

Step 1 (initialization) Generate an initial solution $\sigma^{(0)} = \sigma_0$ (the details of the generation algorithm will be described later), set $d := 0$, $k := 0$ and $T := \emptyset$. Let σ' be the best feasible solution found during this stage of generating σ_0. Set $\sigma_{best} := \sigma'$ and $best := cost(\sigma')$; if no feasible solution was found, set $best := \infty$.

Step 2 (variable depth search) Repeat the following two steps, Step 2-a and Step 2-b, to generate a sequence of solutions $\sigma^{(0)}, \sigma^{(1)}, \ldots, \sigma^{(\bar{d})}$ as long as it does not exit to Step 3.

a (shift move): If $d = 0$ and $N'_{shift}(\sigma^{(0)}) \backslash T = \emptyset$, or if $d \geq 1$ and $N'_{shift}(\sigma^{(d)}) = \emptyset$, then set $\bar{d} := d$ and exit to Step 3. Otherwise choose the solution σ with the minimum $pcost(\sigma)$ in $N'_{shift}(\sigma^{(0)}) \backslash T$ if $d = 0$, or in $N'_{shift}(\sigma^{(d)})$ if $d \geq 1$, and set $\sigma_{k+1} := \sigma$, and $k := k + 1$.

b (swap move): If σ_k is feasible and $cost(\sigma_k) < best$, then set $\sigma_{best} := \sigma_k$, $best := cost(\sigma_k)$, $d^* := \bar{d} := d + 1$ and exit to Step 3. If $pcost(\sigma) \geq pcost(\sigma_k)$ for all σ in $N^{dsj}_{swap}(\sigma_k, I_k)$, then set $\sigma^{(d+1)} := \sigma_k$, $d := d + 1$ and return to Step 2-a. Otherwise choose a solution σ in $N^{dsj}_{swap}(\sigma_k, I_k)$, set $\sigma_{k+1} := \sigma$, $k := k + 1$ and return to Step 2-b. (The details of how to choose σ will be explained later.)

Step 3 (starting solution for the next variable depth search) If the incumbent solution σ_{best} has been updated during Step 2, set $\sigma^{(d^*)} := \sigma_{best}$; otherwise the solution $\sigma^{(d^*)}$ that minimizes $pcost(\sigma)$ among $\sigma^{(0)}, \sigma^{(1)}, \ldots, \sigma^{(\bar{d})}$ is chosen.

Step 4 (exit or go to the next variable depth search) If $d^* = 0$, then set $T := T \cup \{\sigma_1\}$; otherwise set $T := \emptyset$. If $N'_{shift}(\sigma^{(0)}) \backslash T = \emptyset$, go to Step 5; otherwise set $\sigma^{(0)} := \sigma_0 := \sigma^{(d^*)}$, $d := 0$, $k := 0$, and return to Step 2. (If $d^* = 0$ holds, it means that no improvement was attained in the previous variable depth search, and we try another sequence of solutions from the same $\sigma^{(0)}$. In this case, set T is augmented by σ_1 to choose a different solution for the new sequence. If $d^* \geq 1$, on the other hand, no restriction is imposed in the selection of the new sequence.)

Step 5 (halt or random restart) If Steps 1 through 4 are repeated r (prespecified integer) times, or the computational time exceeds τ (prespecified parameter) seconds, output σ_{best} and stop; otherwise return to Step 1.

Modification of the neighborhoods

In algorithm VDS, we use two neighborhoods, $N_{shift}(\sigma_k)$ or $N_{swap}(\sigma_k)$. To reduce computational effort and to improve performance, however, we modify these to $N'_{shift}(\sigma_k)$ or $N^{dsj}_{swap}(\sigma_k, I_k)$, respectively, defined as follows.

Modified shift neighborhood $N'_{shift}(\sigma_k)$: Let $J(\sigma, \sigma') := \{j \in J |\ \sigma(j) \neq \sigma'(j)\}$ be the set of jobs whose assignments were changed in the move from σ to σ', and let $J_k := J(\sigma_{k-1}, \sigma_k)$ $(k \geq 1)$. Note that $|J_k| = 1$ (resp., 2) if the move $\sigma_{k-1} \rightarrow \sigma_k$ is a shift move (resp., a swap move). Then $\tilde{J}_k := \cup_{1 \leq l \leq k, |J_l|=1} J_l$ is the set of jobs involved in the past shift moves in the sequence of $\sigma_0, \sigma_1, \ldots, \sigma_k$. If $k \geq 1$, in defining $N'_{shift}(\sigma_k)$, we forbid changing the assignment of jobs in $\tilde{J}_k$. This restriction is added to avoid cycling among the assignments in the sequence $\sigma_0, \sigma_1, \ldots, \sigma_{k+1}$. We also forbid a move from σ_k to a $\sigma \in N_{shift}(\sigma_k)$ unless

$$pcost(\sigma) - \{p_{\sigma(j)}(\sigma) - p_{\sigma(j)}(\sigma_k)\} < pcost(\sigma_{l^*}), \tag{1.2}$$

holds, where $J(\sigma_k, \sigma) = \{j\}$, and

$$pcost(\sigma_{l^*}) = \min_{0 \le l \le k} pcost(\sigma_l).$$

This is because the assignment σ not satisfying (1.2) has little chance of yielding an assignment better than σ_{l^*}, for the reason as described in Section 7.1. Let N'_{shift} denote the shift neighborhood resulting from the above rules.

Modified swap neighborhood $N^{dsj}_{swap}(\sigma_k, I_k)$: Let $I(\sigma, \sigma') := \{\sigma(j), \sigma'(j) | \ j \in J(\sigma, \sigma')\}$ be the set of agents which participate in the move from σ to σ', and let $I_k := I(\sigma_{k-1}, \sigma_k)$. We shrink $N_{swap}(\sigma_k)$ into $N'_{swap}(\sigma_k, I_k)$, where $N'_{swap}(\sigma_k, I_k)$ consists of those swaps in which at least one of the pertaining jobs is taken from an agent in I_k.

Swap neighborhood is further modified as follows. We call two swap moves from σ to σ' and from σ to σ'' are *disjoint* if $I(\sigma, \sigma') \cap I(\sigma, \sigma'') = \emptyset$. A set of swap moves is called *mutually disjoint* if every pair of swap moves are disjoint. The modified swap neighborhood $N^{dsj}_{swap}(\sigma_k, I_k)$ then consists of those solutions obtainable by simultaneously applying a set of mutually disjoint swap moves in $N'_{swap}(\sigma_k, I_k)$.

Details of Steps 1 and 2

Now we will explain the details of our algorithm VDS.

Generation of an initial solution in Step 1: First, generate randomly an assignment σ. Find a locally optimal solution σ'' by applying $LS(N_{shift}, pcost, \sigma)$, and then find another locally optimal solution σ_0 by $LS(N_{swap}, pcost, \sigma'')$. This σ_0 is used as the initial solution of Step 1.

Choice of the solution σ in Step 2-b: To choose σ_{k+1} from $N^{dsj}_{swap}(\sigma_k, I_k)$, we generate a sequence of solutions $\sigma'_1, \sigma'_2, \ldots, \sigma'_L$ realized by disjoint swap moves in $N'_{swap}(\sigma_k, I_k)$, by the following algorithm.

Algorithm DS

Step 1 Let $\sigma'_0 := \sigma_k$, $S_0 := \{\sigma \in N'_{swap}(\sigma'_0, I_k) | \ pcost(\sigma) < pcost(\sigma_k)\}$, and $l := 1$.

Step 2 Choose the $\sigma \in S_{l-1}$ that minimizes $pcost(\sigma)$, and set $\sigma'_l := \sigma$.

Step 3 Set $S_l := \{\sigma \in N'_{swap}(\sigma'_l, I_k) | \ pcost(\sigma) < pcost(\sigma'_l)$ and $I(\sigma'_l, \sigma) \cap (\cup_{1 \le h \le l} I(\sigma'_{h-1}, \sigma'_h)) = \emptyset\}$. If $S_l = \emptyset$, set $L := l$ and exit; otherwise let $l := l + 1$ and return to Step 2.

Then σ_{k+1} is defined to be the σ'_L when it exits from Step 3. In this case, we set $I_{k+1} := \cup_{1 \le l \le L} I(\sigma'_{l-1}, \sigma'_l)$, for which $|I_{k+1}| = 2L$ holds. See Section 7.2 for discussions about the computational time of Algorithm DS.

5 Computational Results

Our algorithm in Section 4, and four of the known algorithms were run on a workstation Sun Ultra 2 Model 2200 (two 200MHz processors), where the computation

was executed on only a single processor. We tested the benchmark instances called types C and D with up to $n = 200$ and $m = 20$, found in [2]. We also tested the instances called type E, found in [5]. Types C and D instances were taken from OR-Library (`http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html`), and type E instances were generated by ourselves, which are available at `http://www.kuamp.kyoto-u.ac.jp/labs/or/members/yagiura/gap/`.

We first examined the effect of parameter α of Algorithm VDS (α is included in *pcost* defined in (1.1)). Table 5.1 shows the average cost of 10 runs of Algorithm VDS with $r = 1$ and $\tau = \infty$. The column '#feasible' shows the number of runs in which feasible solutions are found. Average computational time for each run is also shown. We can observe that the performance of Algorithm VDS is quite sensitive on α.

We then tested (1) our VDS algorithm (denoted VDS), (2) random multi-start local search (denoted MLS), (3) VDS by Racer and Amini [11] (denoted R&A), (4) tabu search by Laguna et al. [5] (denoted TABU), (5) tabu search for the constraint satisfaction problem by Nonobe and Ibaraki [9] (denoted TABU-CSP). (An improved version of algorithm R&A is proposed in [1]; however, the new version is not tested here, since the idea of [1] is to start the search from good initial solutions generated by the heuristics of Martello and Toth [7, 8], which is also applicable to all other algorithms tested in this paper.) VDS, MLS and R&A are coded in C language by ourselves, while the codes of TABU and TABU-CSP are sent from the authors (which are also written in C). The parameter α in (1.1), used in VDS and MLS, was set to $\alpha = 2$ for types C and D instances, and to $\alpha = 20$ for type E instances. For MLS, $N_{shift} \cup N_{swap}$ is used for the neighborhood. R&A does not include any parameter. The parameters for TABU and TABU-CSP are set to the default values. All algorithms are iterated until their computational time for each problem instance exceeds the prespecified values in Table 5.2 (e.g., for VDS, parameter r is set to ∞ and τ is set to the values in the table).

For comparison purposes, we also include in Table 5.2 the results of the genetic algorithm cited in [2] by Chu and Beasley (denoted GA), where the computational time, reported in [2], is on a different workstation Silicon Graphics Indigo (R4000, 100MHz). The computational time allowed for algorithms VDS, MLS, R&A, TABU and TABU-CSP are set to the half of the time allowed for GA, taking into consideration rough estimation of the power of computers in use. The results of GA for type E instances are not available and denoted as 'N.A.' in the table.

Table 5.3 shows the best costs obtained by the tested algorithms, where '—' means that the algorithm could not obtain a feasible solution. There are five types of problems named types A, B, C, D and E. Among them, types A and B are very easy and there are not much differences among the tested algorithms; hence, only the results of types C, D and E are shown. It is known that types D and E are much more difficult than type C. In the table, '$*$' marks the best solution among those obtained by the six algorithms. From Table 5.3, we can observe that the solution quality of the proposed VDS is better than other five algorithms in most cases.

We have also compared our VDS with the tabu search by Osman [10] for type C instances with up to $n = 60$ [13]. The results indicate that VDS obtains solutions of comparable or even better quality within reasonable amount of computational time. It should be noted, however, that rigorous comparison is not possible, since the computers used for the two algorithms are different.

Table 5.1 : The effect of parameter α.

type	n	m	α	avr. cost	#feasible	avr. time (secs.)
C	100	10	0.25	—	0	0.3
C	100	10	0.50	—	0	1.2
C	100	10	1.00	1429.3	4	0.5
C	100	10	1.50	1423.9	10	0.4
C	100	10	2.00	1409.3	10	0.3
C	100	10	4.00	1414.5	10	0.2
C	200	10	0.25	—	0	10.5
C	200	10	0.50	—	0	7.5
C	200	10	1.00	2817.1	10	5.1
C	200	10	1.50	2816.2	10	2.9
C	200	10	2.00	2819.4	10	2.4
C	200	10	4.00	2821.1	10	2.5
D	100	10	0.25	—	0	2.6
D	100	10	0.50	—	0	9.6
D	100	10	1.00	—	0	57.7
D	100	10	1.50	6415.6	10	70.5
D	100	10	2.00	6394.6	10	47.8
D	100	10	4.00	6414.1	10	21.1
D	200	10	0.25	—	0	34.5
D	200	10	0.50	—	0	114.2
D	200	10	1.00	—	0	1010.1
D	200	10	1.50	12484.8	10	891.3
D	200	10	2.00	12495.7	10	584.4
D	200	10	4.00	12525.4	10	279.7
E	100	10	10.00	—	0	34.4
E	100	10	12.00	—	0	46.8
E	100	10	14.00	—	0	40.2
E	100	10	16.00	11600.1	10	74.9
E	100	10	18.00	11590.5	10	63.1
E	100	10	20.00	11600.8	10	33.8
E	100	10	40.00	11656.5	10	9.2
E	200	10	10.00	—	0	640.9
E	200	10	12.00	—	0	741.2
E	200	10	14.00	—	0	670.9
E	200	10	16.00	23319.6	10	1116.5
E	200	10	18.00	23334.4	10	452.9
E	200	10	20.00	23346.8	10	226.6
E	200	10	40.00	23428.3	10	87.7

Table 5.2: The computational times (secs.) of the six algorithms. (Note that the computational time of GA is on a different computer.)

type	n	m	others†	GA‡
C	100	5	151.2	302.4
C	100	10	197.1	394.2
C	100	20	334.6	669.3
C	200	5	405.0	810.1
C	200	10	523.0	1046.0
C	200	20	896.1	1792.3
D	100	5	265.1	530.3
D	100	10	547.3	1094.7
D	100	20	1063.0	2126.1
D	200	5	971.4	1942.8
D	200	10	1594.8	3189.6
D	200	20	2782.5	5565.1
E	100	5	300.0	N.A.
E	100	10	600.0	N.A.
E	100	20	900.0	N.A.
E	200	5	1000.0	N.A.
E	200	10	2000.0	N.A.
E	200	20	3000.0	N.A.

†Computational time on Sun Ultra 2 Model 2200 (200MHz) allowed for algorithms VDS, MLS, R&A, TABU and TABU-CSP.

‡Computational time on Silicon Graphics Indigo (100MHz) allowed for algorithm GA.

Table 5.3 : The best costs of the six algorithms.

type	n	m	VDS	MLS	R&A	TABU	TABU-CSP	GA
C	100	5	*1931	*1931	1938	*1931	*1931	*1931
C	100	10	*1402	1412	1405	1403	1403	1403
C	100	20	1246	1263	1250	*1244	1245	*1244
C	200	5	*3457	3463	3469	*3457	3459	3458
C	200	10	*2809	2839	2835	2812	2817	2814
C	200	20	2400	2432	2413	*2396	2400	2397
D	100	5	*6365	6413	—	6386	6415	6373
D	100	10	6380	6476	6532	6398	6456	*6379
D	100	20	*6266	6396	6396	6283	6315	6269
D	200	5	*12778	12851	—	12788	12915	12796
D	200	10	*12496	12664	12707	12519	12777	12601
D	200	20	*12323	12628	12642	12416	12596	12452
E	100	5	*12685	12722	12917	12687††	12699	N.A.
E	100	10	*11582	11700	12047	11650††	11689	N.A.
E	100	20	*8483	8816	8883	8555††	8658	N.A.
E	200	5	*24946	24969	25649	25203††	24958	N.A.
E	200	10	*23334	23498	24710	23567††	23425	N.A.
E	200	20	*22439	22831	23947	22735††	22584	N.A.

††Results after 20,000 seconds on Sun Ultra 2 Model 2200 (200MHz).

6 Conclusion and Discussion

We proposed a variable depth search algorithm for the generalized assignment problem. Our algorithm features: (1) the search into the infeasible region, and (2) an adaptive use of two different neighborhoods, shift and swap. Comparing our algorithm with other metaheuristic algorithms, it was observed that our algorithm was able to obtain solutions of better quality for types D and E instances, which is known to be very hard.

Let us call $\bar{d} - d^*$ ($\bar{d}$ and d^* are indices of the solutions in Step 3 of VDS) *overshoot*. This indicates in some sense the number of unnecessary moves. It is observed that the overshoot becomes quite large for types D and E instances compared to other types. For example, the average number of overshoots for an instance of type D with $n = 200$ and $m = 20$ is 18.9, while that for an instance of type C with the same size is 3.0. Some additional rules to reduce this overshoot may be useful to improve the performance of VDS.

It is also observed that the performance of our VDS is sensitive to the parameter α (the weight on the penalty term). It may also be important to incorporate the ability of tuning α automatically in our algorithm.

Acknowledgement

The authors are grateful to Fred Glover and Manuel Laguna [5], and Koji Nonobe [9] for providing us their original codes.

Bibliography

[1] M.M. Amini and M. Racer, "A Hybrid Heuristic for the Generalized Assignment Problem," *European J. Oper. Res.*, 87 (1995) 343–348.

[2] P.C. Chu and J.E. Beasley, "A Genetic Algorithm for the Generalized Assignment Problem," *Computers Oper. Res.*, 24 (1997) 17–23.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (Freeman, 1979).

[4] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Syst. Tech. J.*, 49 (1970) 291–307.

[5] M. Laguna, J.P. Kelly, J.L. González-Velarde and F. Glover, "Tabu Search for the Multilevel Generalized Assignment Problem," *European J. Oper. Res.*, 82 (1995) 176–189.

[6] S. Lin and B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Oper. Res.*, 21 (1973) 498–516.

[7] S. Martello and P. Toth, "An Algorithm for the Generalized Assignment Problem," *Proc. IFORS*, (1981) 589–603.

[8] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, (John Wiley & Sons, 1990).

[9] K. Nonobe and T. Ibaraki, "A Tabu Search Approach to the CSP (Constraint Satisfaction Problem) as a General Problem Solver," *European J. Oper. Res., Special Issue on Tabu Search*, (to appear).

[10] I.H. Osman, "Heuristics for the Generalized Assignment Problem: Simulated Annealing and Tabu Search Approaches," *OR Spektrum*, 17 (1995) 211–225.

[11] M. Racer and M.M. Amini, "A Robust Heuristic for the Generalized Assignment Problem," *Ann. Oper. Res.*, 50 (1994) 487–503.

[12] S. Sahni and T. Gonzalez, "P-Complete Approximation Problems," *J. ACM*, 23 (1976) 555–565.

[13] T. Yamaguchi, "A Variable Depth Search for the Generalized Assignment Problem (in Japanese)," Graduate Thesis, Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, 1996.

7 Appendices

7.1 Remark on Rule (1.2)

The rule (1.2) is based on the following observation. Consider two solutions σ_k and σ' (σ' is not necessarily in $N'_{shift}(\sigma_k)$). Then σ' can be obtained from σ_k by the set of shift moves $\sigma_k(j) \rightarrow \sigma'(j)$ for all jobs j in $J(\sigma_k, \sigma')$. Note that a swap move can be represented by two shift moves. Then

$$\sum_{j \in J(\sigma_k, \sigma')} \left\{ c_{\sigma'(j),j} - c_{\sigma_k(j),j} - \alpha \min \left\{ p_{\sigma_k(j)}(\sigma_k), a_{\sigma_k(j),j} \right\} \right\} \leq pcost(\sigma') - pcost(\sigma_k) \tag{1.3}$$

holds, since

$$\begin{aligned} p_i(\sigma') &\geq p_i(\sigma_k) - \min \left\{ p_i(\sigma_k), \sum_{j \in J(\sigma_k, \sigma'), \sigma_k(j) = i} a_{ij} \right\} \\ &\geq p_i(\sigma_k) - \sum_{j \in J(\sigma_k, \sigma'), \sigma_k(j) = i} \min \left\{ p_i(\sigma_k), a_{ij} \right\} \end{aligned}$$

for all $i \in I$. Each term in the left side of (1.3) denotes the increase of *pcost* by the shift $\sigma_k(j) \rightarrow \sigma'(j)$ of job j, in which the increase of $p_{\sigma'(j)}$ is neglected. Therefore, if an assignment σ' satisfying $pcost(\sigma') < pcost(\sigma_k)$ can be obtained by a sequence of moves, then there exists a first move $\sigma_k \rightarrow \sigma$ such that $\sigma \in N_{shift}(\sigma_k)$, $\sigma(j) = \sigma'(j)$ for $\{j\} = J(\sigma_k, \sigma)$ and

$$c_{\sigma(j),j} - c_{\sigma_k(j),j} - \alpha \min \left\{ p_{\sigma_k(j)}(\sigma_k), a_{\sigma_k(j),j} \right\} < 0.$$

The last condition is equivalent to

$$pcost(\sigma) - \left\{ p_{\sigma(j)}(\sigma) - p_{\sigma(j)}(\sigma_k) \right\} < pcost(\sigma_k), \tag{1.4}$$

since

$$\begin{aligned} & pcost(\sigma) - pcost(\sigma_k) \\ & = c_{\sigma(j),j} - c_{\sigma_k(j),j} - \alpha \min\left\{p_{\sigma_k(j)}(\sigma_k), a_{\sigma_k(j),j}\right\} + \left\{p_{\sigma(j)}(\sigma) - p_{\sigma(j)}(\sigma_k)\right\} . \end{aligned}$$

Therefore, in considering the first shift move from σ_k, we restrict only to those σ satisfying (1.4), expecting that the subsequent swap moves from σ will eventually find an assignment σ' satisfying $pcost(\sigma') < pcost(\sigma_k)$ if such σ' exists.

In our rule of defining N'_{shift}, we further strengthen (1.4) to (1.2) by replacing the right-hand side by $pcost(\sigma_{l^*})$ ($\leq pcost(\sigma_k)$). This is because we want to find a solution σ' better than σ_{l^*} in our search in Step 2 of VDS.

7.2 Remark on Algorithm DS

Let us consider the computational time needed to choose the σ_{k+1} by Algorithm DS. We store S_l by the set $\{CH(\sigma'_l,\sigma)|\ \sigma \in S_l\}$, where $CH(\sigma'_l,\sigma) := \{(j;\sigma'_l(j) \rightarrow \sigma(j))|\ j \in J(\sigma'_l,\sigma)\}$. This is the set of changes of the assignments of the two jobs in $J(\sigma'_l,\sigma)$ involved in the swap moves $\sigma'_l \rightarrow \sigma$ of S_l. Then $\{CH(\sigma'_l,\sigma)|\ \sigma \in S_l\} \subset \{CH(\sigma'_{l-1},\sigma)|\ \sigma \in S_{l-1}\}$ holds for all l. Let $\Delta pcost(CH(\sigma'_l,\sigma)) := pcost(\sigma) - pcost(\sigma'_l)$. Note that $\Delta pcost$ for the changes in $\{CH(\sigma'_l,\sigma)|\ \sigma \in S_l\}$ is independent of l during the execution of DS, since the swap moves remaining in S_l are disjoint to those previously used. Therefore, $\Delta pcost$ for all swap moves in S_l, $1 \leq l \leq L$, were already computed in Step 1 when S_0 was prepared in $O(|N'_{swap}(\sigma'_0, I_k)|)$ time. The time for Step 2 is clearly $O(|S_{l-1}|)$. The update of S_l in Step 3 is also possible in $O(|S_{l-1}|)$ time since it only requires to delete from $\{CH(\sigma'_{l-1},\sigma)|\ \sigma \in S_{l-1}\}$ the changes $CH(\sigma'_{l-1},\sigma)$ of swap moves $\sigma'_{l-1} \rightarrow \sigma$ such that $I(\sigma'_{l-1},\sigma) \cap I(\sigma'_{l-1},\sigma'_l) \neq \emptyset$. Therefore, the computational time of DS is $O(|N'_{swap}(\sigma_k, I_k)| + \sum_{l=0}^{L-1} |S_l|) = O(|N'_{swap}(\sigma_k, I_k)| + L|S_0|)$, since $|S_l|$ is monotonically decreasing with l.

On the other hand, the computational time would become $O(L|N'_{swap}(\sigma_k, I_k)|)$ if we just choose the best solution in $N'_{swap}(\sigma'_l, I_k)$ L times. Usually, $|S_0|$ is much smaller than $|N'_{swap}(\sigma_k, I_k)|$, and the algorithm DS thus reduces computational time.