

求解卸装一体化的车辆路径问题的混合启发式算法

陈 萍 黄厚宽 董兴业

(北京交通大学计算机科学与技术学院 北京 100044)

摘 要 提出一种结合蚁群系统(Ant Colony System, ACS)和变邻域下降搜索(Variable Neighborhood Descent, VND)的混合启发式算法 ACS_VND, 求解卸装一体化车辆路径问题. 利用基于插入的 ACS 解构造方法产生多个弱可行解, 再逐个转换成强可行解, 并选择其中最好的作为 VND 的初始解. 在 VND 过程中使用三种不同的邻域结构: 插入、交换和 2-opt 依次对解进行迭代优化. 对 55 个规模为 22~199 的 benchmark 算例的求解结果表明, 算法 ACS_VND 能在较短时间内获得 52 个算例的已知最好解, 并且更新了其中 44 个算例的已知最好解, 求解性能优于现有算法.

关键词 卸装一体化车辆路径问题; 混合启发式算法; 蚁群系统; 变邻域下降搜索; 组合优化; NP 难
中图法分类号 TP301

A Hybrid Heuristic Algorithm for the Vehicle Routing Problem with Simultaneous Delivery and Pickup

CHEN Ping HUANG Hou-Kuan DONG Xing-Ye

(School of Computer Science and Information Technology, Beijing Jiaotong University, Beijing 100044)

Abstract A hybrid heuristic algorithm ACS_VND is proposed and applied to solving the vehicle routing problem with simultaneous delivery and pickup, which combines two metaheuristics, i. e., Ant Colony System and Variable Neighborhood Descent. Several weakly feasible solutions are built by an insertion based ACS solution construction method, which are then transformed into strongly feasible ones, the best of which is taken as the initial solution of the VND procedure. During the VND procedure, three different neighborhood structures, i. e., insertion, swap and 2-opt are successively used. Computational results on the 55 benchmark problems with the size ranging from 22 to 199, show that the proposed hybrid heuristic algorithm can find the best known solution for 52 problems in a short time; furthermore, the best known solutions for 44 problems are updated, which indicates that the proposed ACS_VND outperforms other algorithms in literatures.

Keywords vehicle routing problem with simultaneous delivery and pickup; hybrid metaheuristics; ant colony system; variable neighborhood descent; combinatorial optimization; NP-hard

1 引 言

车辆路径问题(Vehicle Routing Problem, VRP)

一直是运筹学、管理学、计算机科学等领域的研究热点问题, 在现实生活中有着广泛的应用, 如物流配送、邮政投递、校车路径安排、铁路和飞机的调度等, 是一个具有重要理论意义和实际应用价值的研究

收稿日期: 2007-11-26. 本课题得到国家“九七三”重点基础研究发展规划项目基金(2006CB705500)资助. 陈 萍, 女, 1981 年生, 博士研究生, 主要研究方向为车辆路径优化算法、元启发式以及组合优化. E-mail: chenpingbjtu@gmail.com. 黄厚宽, 男, 1940 年生, 教授, 博士生导师, 主要研究领域为人工智能、机器学习、数据仓库、数据挖掘、决策支持系统以及多智能体系统等. 董兴业, 男, 1974 年生, 博士研究生, 主要研究方向为调度理论和算法、组合优化.

课题. 近年来, 逆向物流的发展使得 VRP 中另一形式的问题——卸装一体化车辆路径问题 (Vehicle Routing Problem with Simultaneous Delivery and Pickup, VRPSDP) 引起研究者的关注. 卸装一体化车辆路径问题可简单描述如下: 安排车辆为一定数量的客户送货, 并从客户处收集一定数量的物品 (如货物包装材料、垃圾等). 每个客户的地理位置以及卸货、装货需求事先已知. 要求在满足一定约束条件 (如车辆容量限制等) 的前提下, 安排车辆满足客户需求使得总的车辆行程长度最短.

VRP 自 1959 年由 Dantzig^[1] 提出后, 几十年来已取得大量研究成果, 然而关于 VRPSDP 的研究却非常少. 1989 年 Min^[2] 针对一个图书馆配送的实例首次定义了该问题. 以后的十年中, 几乎没有关于该问题的讨论. 逆向物流的兴起使得研究者重新开始对该问题进行研究和探讨. 文献[2-5]讨论了该问题的若干应用实例. 由于 VRP 是 NP 难问题, 因此, VRPSDP 也是 NP 难的, 而且比基本 VRP 问题更复杂, 因此, 对此类问题的求解方法研究主要集中在能在较短时间内给出较优解的启发式算法 (heuristic) 和元启发式算法 (metaheuristic), 现已取得了一定的研究成果, 主要有构造性算法, 如文献[6]中 Dethloff 提出的带有参数的插入法, 文献[7]中的先聚类后插入算法; 禁忌搜索, 如文献[8]中 Crispim 等提出的基于禁忌搜索的混合算法, 文献[9]中 Tang Montané 等提出的基于四种邻域结构和两种搜索策略的禁忌搜索, 文献[10]中 Chen 和 Wu 提出一种基于“记录”和禁忌表的混合启发式算法. 最近, Bianchessi 和 Righini 提出一些构造性算法、局部搜索算法以及禁忌搜索算法并加以比较, 其中重点讨论了基于复合多邻域结构的禁忌搜索算法^[11]. Tang Montané 等^[9]给出一组测试算例解的下界, 这些下界表明现有算法虽能在一定程度上解决 VRPSDP, 但求解质量还有较大改进空间.

蚁群优化 (Ant Colony Optimization, ACO) 是一种基于群体的元启发式算法, 最初的灵感来源于真实蚂蚁搜寻食物的行为^[12]——以信息素作为媒介间接进行信息交换. 目前蚁群优化已经成功应用于多个 NP 难的组合优化问题求解. 变邻域搜索 (Variable Neighborhood Search, VNS) 最早由 Hansen 和 Mladenovic^[13-14] 提出, 其核心思想是: 邻域结构定义了搜索空间的拓扑特性, 不同的邻域结构对应搜索空间的不同区域. 一般地, 问题解空间中某个区域的特性不同于其它区域, 因此, 动态使用不

同的邻域结构能够增加解的多样性. 变邻域下降 (Variable Neighborhood Descent, VND) 是 VNS 的一种变形, 它通过一种确定的方式来改变邻域结构的使用. 根据文献[14]中对元启发式算法的分类, 蚁群优化属于基于群体的算法, 而变邻域下降搜索则是属于轨迹法. 基于群体的元启发式算法的优势在于善于发现搜索空间中可能存在最优解的区域, 而轨迹法的优势在于善于探索搜索空间中较好的区域. 因此, 将二者结合可以充分利用各自的优势, 提高算法的搜索性能和效率. 本文将蚁群优化中的一种——蚁群系统 (Ant Colony System, ACS) 与 VND 相结合, 提出一种混合启发式算法 ACS_VND 用于求解 VRPSDP. 利用 55 个 benchmark 测试算例进行实验, 并与文献[2]以及文献[6-11]中的算法求解结果比较, 验证本文算法的有效性.

2 卸装一体化车辆路径问题描述

2.1 问题描述

本文利用有向带权图 G 描述卸装一体化车辆路径问题: 假设 $G = (V, A, C)$, 其中, $V = \{i | i = 0, 1, \dots, n\}$ 是顶点集 (0 表示配送中心, 其它表示客户); $A = \{(i, j) | i, j \in V\}$ 是连接各顶点的弧集; $C = \{c_{ij} | (i, j) \in A\}$ 是权重矩阵, c_{ij} 表示从客户 i 到客户 j 的距离. 任意客户 $i (i = 1, 2, \dots, n)$ 都有一定卸货需求 d_i 与装货需求 p_i . 安排车辆为所有客户服务 (假设所有车辆为同一车型, 车辆最大容量为 Q), 要求满足以下条件并使得车辆总行程长度最短:

- (a) 每辆车都从仓库出发, 并最终返回仓库;
- (b) 每个客户都只被一辆车服务, 且仅被服务一次;
- (c) 任一车辆在行程过程中, 载重始终不能超过 Q .

2.2 解的可行性

设 $s = \{r_i | i = \{1, 2, \dots, k\}\}$ 是 VRPSDP 问题的一个解, 其中 r_i 对应一条车辆路径. 由 2.1 节中的 VRPSDP 问题描述可知, s 是 VRPSDP 问题的一个可行解的充分必要条件是: 对任意 r_i , 都满足

- (1) r_i 上所有客户的总的卸货需求 $D(x)$ 不超过 Q ;
- (2) r_i 上所有客户的总的装货需求 $P(x)$ 不超过 Q ;
- (3) 车辆访问 r_i 上任何客户之后载重都不超过 Q .

若 $\forall r_i \in s$ 都满足条件(1)(2)(3), 则称 s 满足强可行性条件, 是强可行解; 若 $\forall r_i \in s$ 都满足条件(1)(2), 但 $\exists r_i \in s$ 不满足条件(3), 则称 s 满足弱可行性条件, 是弱可行解. 由于 Mosheiov^[3] 已经证明, 如果 $D(x)$ 和 $P(x)$ 都不超过车辆容量限制, 则 r_i 一定可以通过某种转化成为可行路径. 因此, 若 s 是弱可行解, 则一定可以通过某种方式转化为强可行解.

3 混合启发式算法 ACS_VND

3.1 信息素初始化

首先使用最近邻启发式(Nearest Neighborhood Heuristic, NNH)构造一个强可行解 s_0 , 并根据式(1)设定信息素初值.

$$\tau_0 = 1/n \cdot f(s_0) \quad (1)$$

其中, n 是客户数量.

NNH 构造解的步骤如下:

1. 从尚未访问的客户节点中, 选择距离配送中心最小的客户, 开始一条新的车辆路径 r ;
2. 若 V_0 不为空, 则从中选择距离 r 上最后一个客户最近的客户, 作为下一个访问的节点; 否则, 转步 1, 直到所有客户都已被访问. 这里, 将 V_0 定义为尚未被访问, 且加入 r 后, 使得 r 仍能约束强可行性条件的所有客户节点的集合.

3.2 构建可行解

由于弱可行性条件检查比较简单, 在算法 ACS_VND 的构建解阶段, 首先产生一组弱可行解, 然后转化成强可行解. 在 ACS 解的构造方式的基础上, 算法 ACS_VND 中使用一种基于插入的启发式方法构造弱可行解. 首先, 从配送中心 0 出发, 随机选择一个客户, 开始一条新的路径 r ; 然后, 根据伪随机比例规则(2)(3), 从 V_1 中选择客户 k 并将它插入到当前路径 r 上节点 i 与 j 之间, 这里, V_1 是指满足以下条件的客户节点 k 的集合: k 尚未被访问, 且若 k 加入 r 仍能保证 r 满足弱可行性条件. ω_k 的定义如式(4), 它决定了客户 k 被选中的可能性大小, 其中, i 和 j 是当前路径 r 上的两个相邻的节点. τ'_{ijk} 的定义如式(5), τ_{ik} 和 τ_{kj} 是插入 k 后新增的两条弧 (i, k) 和 (k, j) 上的信息素, τ_{ij} 是删去的弧 (i, j) 上的信息素. η_{ijk} 是启发式信息, 定义如式(6), 其中第一项考虑了客户与配送中心之间的距离, 以避免距离仓库较远的客户插入得较晚, 从而增加额外的代价; 第二项表示将客户 k 插入到当前路径上客户 i 与客户 j 之间时增加的路径长度. α, β 是 τ'_{ijk} 和 η_{ijk} 的相对影响因子. 不断地从 V_1 中选择客户, 直到 V_1

为空, 结束当前路径 r 的构造; 若所有客户都已在当前解中, 结束算法; 否则, 重新开始一条新的 r 并重复上述构造过程. 为取得利用历史信息 and 随机选择之间的平衡, 算法 ACS_VND 中动态调整 q_0 的大小, 使其取值为 q_{\min} 或 q_{\max} .

$$k = \begin{cases} \arg \max_{h \in V_1} \{\omega_h\}, & q \leq q_0 \\ S, & \text{其他} \end{cases} \quad (2)$$

$$S: p_k = \begin{cases} \frac{\omega_k}{\sum_{h \in V_1} \omega_h}, & k \in V_1 \\ 0, & \text{其他} \end{cases} \quad (3)$$

$$\omega_k = \max_{(i,j) \in r} \{[\tau'_{ijk}]^\alpha [\eta_{ijk}]^\beta\} \quad (4)$$

$$\tau'_{ijk} = (\tau_{ik} + \tau_{kj}) / 2\tau_{ij} \quad (5)$$

$$\eta_{ijk} = c_{0k} - (c_{ik} + c_{kj} - c_{ij}) \quad (6)$$

算法 ACS_VND 中利用与文献[10]相同的方法将弱可行解转化成强可行解, 即从头至尾逐个扫描每一条路径 r 上的客户, 若访问当前客户后 r 不能满足强可行性条件, 则跳过当前客户扫描下一个; 否则, 继续扫描下一个; 最后, 按照逆序将在第一次扫描中被跳过的客户逐个重新加入 r , 即可得到一个强可行解.

3.3 局部信息素更新

根据式(7), 利用构造的每一个解 s 进行局部信息素更新, 其中, $0 < \rho_1 < 1$ 是信息素挥发系数, τ_0 是信息素初值.

$$\tau_{ij} = (1 - \rho_1)\tau_{ij} + \rho_1\tau_0, \quad \tau_{ij} \in s \quad (7)$$

3.4 变邻域下降搜索

变邻域下降搜索的基本步骤是: 从初始解出发, 选择一种邻域结构进行局部搜索, 直到找到局部最优解; 以当前局部最优解为初始解, 使用另一种邻域结构继续进行局部搜索; 当使用任何一种邻域结构都不能继续改进当前解时, 结束 VND 过程.

3.4.1 VND 过程

算法 1. VND 过程.

1. 输入初始解 s , 选择一组邻域结构 $N_k, k = 1, 2, \dots, k_{\max}$; 令 $p = 0$;
2. WHILE $p < k_{\max}$ DO
3. $k = 1$;
4. WHILE $k \leq k_{\max}$ DO
5. 以 s 为初始解, 在 N_k 定义的邻域中进行局部搜索, 直到找到局部最优解 s^* ;
6. IF $f(s^*) < f(s)$ THEN
7. $s = s^*$;
8. $p = 0$;
9. ELSE

10. $p = p + 1$;
11. ENDIF
12. $k = k + 1$;
13. END WHILE
14. END WHILE
15. 输出 s , 算法结束.

3.4.2 邻域结构

在使用变邻域下降搜索之前,需要定义一组邻域结构.算法 ACS_VND 中分别使用 3 种求解 VRP 问题时常用的邻域结构:插入(insert)、交换(swap)和 2-opt.

(1) 插入(insert)

将解 s 中的某个客户 i 从当前位置 p_1 移到 s 的另一个位置 p_2 (p_1 与 p_2 可属于同一路径,也可属于不同路径),产生新解 s' .例如,解 $s = 0 - \boxed{3} - 5 - \underline{7} - 0 - \underline{1} - 2 - 4 - 6 - 0$,将客户 3 从当前的 2 号位置移到 4 号或 6 号位置,产生新解 $s' = 0 - 5 - 7 - \boxed{3} - 0 - 1 - 2 - 4 - 6 - 0$ 或 $s'' = 0 - 5 - 7 - 0 - 1 - \boxed{3} - 2 - 4 - 6 - 0$.当某条路径上没有客户节点时,删去该路径,从而使得车辆个数减少.例如,解 $s = 0 - 1 - 2 - 5 - \boxed{0-0} - 3 - 6 - 4 - 7 - 0$,删去空的子路径后,得到 $s = 0 - 1 - 2 - 5 - \boxed{0} - 3 - 6 - 4 - 7 - 0$.

(2) 交换(swap)

将解 s 中的客户 i 和 j 的位置互换(i 和 j 可属于同一路径,也可属于不同路径),产生新解 s' .例如,解 $s = 0 - \boxed{3} - 5 - \underline{7} - 0 - 1 - 2 - 4 - 6 - 0$,交换同一路径上的客户 3 与 7,产生新解 $s' = 0 - \underline{7} - 5 - \boxed{3} - 0 - 1 - 2 - 4 - 6 - 0$;解 $s = 0 - \boxed{3} - 5 - 7 - 0 - 1 - \underline{2} - 4 - 6 - 0$,交换不同路径上的客户 3 与 2,产生新解 $s'' = 0 - \underline{2} - 5 - 7 - 0 - 1 - \boxed{3} - 4 - 6 - 0$.

(3) 2-opt

解 s 中同一路径上的两个客户 i 和 j ,在解 s 中的位置分别为 p_i 与 p_j ($p_i < p_j$).2-opt 是指将 $p_i + 1$ 位置上的客户与 j 交换,并将 $p_i + 1$ 和客户 j (不包括 $p_i + 1$ 位置上的客户和客户节点 j) 之间的客货节点按逆序访问.例如,解 $s = 0 - \underline{1} - 9 - 5 - 7 - \underline{4} - 0 - \underline{2} - 6 - 3 - 8 - \underline{10} - 11 - 0$,对两条路径分别通过 2-opt 优化后,得到新解 $s' = 0 - 1 - 4 - \underline{7-5} - 9 - 0 - 2 - 10 - \underline{8-3} - 6 - 11 - 0$.

3.4.3 搜索策略

在搜索过程中,只考查满足如下条件的移动(move):至少引入一条弧 (i, j) ,使得 $c_{i,j} \leq \lambda a_i$ 或

$c_{i,j} \leq \lambda a_j$,其中 $a_i = \sum_{j=0}^n c_{ij} / n$, λ 取值为 λ_{\min} 或 λ_{\max} ,从

而能够动态改变考查的邻域空间大小.在使用一种邻域结构局部搜索时只接受最好的邻域解,直到陷入局部最优.实验中发现,按照先插入、再交换最后 2-opt 的顺序,算法性能表现最好,因此,在第 4 节的实验中,均按此顺序使用这 3 种邻域结构.

3.5 全局信息素更新

每次迭代结束后,根据全局信息素更新规则(8)(9),使用历史最优解 s_{sb} 进行全局信息素更新.

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\Delta\tau, \quad \forall (i, j) \in s_{sb} \quad (8)$$

$$\Delta\tau = 1/f(s_{sb}) \quad (9)$$

3.6 算法 ACS_VND

本小节给出整个算法 ACS_VND 的过程.首先,给出相关符号:蚂蚁个数 ant_num ,信息素挥发系数 ρ, ρ_1 , 阈值 q_0 以及相对影响因子 α 和 β ;最大迭代次数 $MaxIter$,最大连续无改进代数 $MaxConsNoImprove$;迭代数计数 $iter$,连续无改进迭代数计数 $cons_no_improve$.

算法 2. ACS_VND 算法伪码.

1. 初始化参数 $ant_num, \rho, \rho_1, q_0, \alpha, \beta, \lambda, MaxIter, MaxConsNoImprove$;令 $iter = 0, cons_no_improve = 0$;本次迭代最优解 s_{ib} , 历史最优解 s_{sb} , 令 $f(s_{sb})$ 为一足够大的正数;
2. 初始化信息素:令 $s_0 \leftarrow NNH()$; $\tau_0 = 1/n \times f(s_0)$;
3. WHILE $iter < MaxIter$ && $cons_no_improve < MaxConsNoImprove$ DO
4. $k = 0$;
5. WHILE $k < ant_num$ DO
6. 根据 3.2 节的方法构造解 s_k ;
7. $k = k + 1$;
8. ENDWHILE
9. 根据式(7)进行局部信息素更新;
10. 从解种群中选出 s_{ib} ,以 s_{ib} 为初始解,执行 VND 过程,步骤同 3.4.1 节中算法 1;
11. IF $f(s_{ib}) < f(s_{sb})$ THEN
12. $s_{sb} = s_{ib}$;
13. $cons_no_improve = 0$;
14. ELSE
15. $cons_no_improve = cons_no_improve + 1$;
16. ENDIF
17. 根据式(8)(9)进行全局信息素更新;
18. $iter = iter + 1$;
19. 更新 λ, q_0 ;
20. ENDWHILE
21. 输出 s_{sb} , 算法结束.

4 实验结果与比较分析

为测试算法 ACS_VND 的性能,我们利用文献中 3 组测试算例进行实验:Min 算例^[2]、Dethloff 算例^[6]以及 Salhi 和 Nagy 算例^[7],分别是图书馆应用实例,随机生成的 VRPSDP 算例以及基本 VRP-benchmark 问题的扩展。

Min 算例是一个图书馆配送的实例. 从一个公共图书馆为其他 22 个分馆发送并收集图书、胶片、磁带等,已知各馆之间的距离以及所有分馆的卸、装货请求,车辆容量限制为 10500,卸货请求总和为 20300,装货请求总和为 19950,因此,至少需要两辆车才能完成配送任务。

Dethloff 算例是文献[6]中随机生成的 40 个算例,问题规模都是 50. 根据地理位置的分布以及最少需要车辆数的不同,可将它们分成四组:SCA3x、SCA8x、CON3x、CON8x. 其中,SCA 算例中的客户均匀分布在区间[0,100];而 CON 算例中一半客户均匀分布在[100/3, 200/3]内,另一半则均匀分布在区间[0,100]. 在[0,100]内随机产生客户的卸货请求 d_i ,并通过 $p_i = (0.5 + \gamma_i) \times d_i$ 产生装货请求 p_i ,其中, γ_i 是[0,1]上的随机数。

Salhi 和 Nagy 将基本 VRP 的 benchmark^[16]加以扩展,作为 VRPSDP 问题的测试算例,问题规模范围是 50~199^[7]. 每个规模都包含两个算例,其中 CMTxX 通过以下变换得到:原 benchmark 中的客

户位置坐标 (x,y) 保持不变;对每一个客户 a ,计算 $r_a = \min(x_a/y_a, y_a/x_a)$,然后分别计算 $d_a = r_a \times t_a$, $p_a = (1 - r_a) \times t_a$,得到该客户的卸、装货需求,其中, t_a 是基本 VRP 的 benchmark 问题中该客户的需求. 而另一个算例 CMTxY,仅是卸、装请求与 CMTxX 相反,其他与 CMTxX 相同。

算法 ACS_VND 用 C++实现,运行在 Pentium IV 2.93GHz 处理器上,内存为 1GB. 通过大量实验,确定参数设置如下: $ant_num=6$, $\rho=0.2$, $\rho_1=0.1$, $q_0=0.9$, $\alpha=\beta=1$; $MaxIter=100+n$, $MaxConsNoImprove=MaxIter/4$; $\lambda_{\min}=0.75$, $\lambda_{\max}=1.50$,每隔 20 代更新 $\lambda=\lambda_{\max}$,其余则令 $\lambda=\lambda_{\min}$; $q_{\min}=0.7$, $q_{\max}=0.9$,当历史最优解连续 5 次无改进时, $q_0=q_{\min}$,否则,令 $q_0=q_{\max}$ 。

4.1 混合算法的性能

为了验证混合启发式算法的性能,我们分别进行 3 组实验. 第 1 组实验只使用 ACS,第 2 组实验只使用 VND,第 3 组实验是本文提出的混合算法 ACS_VND,3 组实验中的其他设置相同. 不失一般性,我们选择 Salhi 和 Nagy 算例作为实验测试数据. 实验结果如表 1 所示,其中第 1 列是算例名称,第 2 列是算例中的客户个数. 每组实验中找到的最好解的路径总长度 L ,车辆个数 k 以及 CPU 时间 t/s 也都在表中列出(L,k 和 t 在本文其它表中的含义均同此表);表 1 中最后一行给出 3 组实验分别求得的结果。

表 1 混合算法 ACS_VND 的性能测试结果

算例	n	ACS 的结果			VND 的结果			ACS_VND 的结果		
		L	k	t^a/s	L	k	t^b/s	L	k	t^b/s
CMT1X	50	505.28	3	0.58	496.63	3	0.05	466.77	3	1.27
CMT1Y	50	488.58	3	0.56	479.25	3	0.05	466.77	3	1.09
CMT2X	75	833.34	6	0.97	754.50	6	0.09	693.50	6	5.11
CMT2Y	75	799.13	6	1.06	710.36	6	0.09	666.75	6	5.46
CMT3X	100	853.13	5	3.36	774.93	5	0.15	715.51	5	11.77
CMT3Y	100	857.46	5	3.41	774.44	5	0.18	724.98	5	12.55
CMT12X	100	823.48	5	3.09	733.35	6	0.15	669.10	5	11.77
CMT12Y	100	844.44	5	2.07	711.39	6	0.15	662.41	5	11.65
CMT11X	120	1055.71	4	9.40	1003.00	4	0.27	842.58	4	26.66
CMT11Y	120	975.19	4	9.29	1009.09	4	0.25	843.28	4	30.27
CMT4X	150	1147.36	7	6.91	888.61	7	0.33	843.24	7	48.95
CMT4Y	150	1147.82	7	7.20	959.27	7	0.38	861.40	7	47.00
CMT5X	199	1527.40	10	15.28	1228.40	10	0.74	1074.88	10	102.71
CMT5Y	199	1494.02	10	13.67	1188.95	10	0.63	1089.88	10	102.83
Avg.		953.74	5.7	5.49	836.58	5.9	0.25	758.60	5.7	29.94

通过表 1 可以看出,混合算法 ACS_VND 求解质量明显优于单独使用 ACS 或 VND,表明将 ACS 和 VND 结合是有效的. 与 ACS 和 VND 相比,

ACS_VND 的时间开销较大,但仍能在较短时间内求得问题较好解,所有问题的求解时间都在 2min 之内。

4.2 与现有算法的比较

为进一步验证本文算法 ACS_VND 的性能,我们将算法 ACS_VND 求得的最好解与现有算法的结果进行比较,如表 2~表 6,其中目前已知最好解用黑体标出.

表 2 是算法 ACS_VND 和现有算法在 Min 算例上的最好解比较. 所有算法求得的最好解的车辆个数都是 2,故不在表中列出. 本文算法求得的最好

表 2 算法 ACS_VND 和现有算法在 Min 算例上的最好解比较

算例	Min ^[2]	Dethloff ^[6]	Tang Montané&.Galvão ^[9]	ACS_VND
MIN22	94	91	88	88

解的路径长度与文献[9]中算法的结果相同,优于文献[2,6]中的算法结果,而且求解时间极短,仅为 0.1s.

表 3 是算法 ACS_VND 和现有算法在 Dethloff 各算例上的最好解比较. 从该表可以看出,在所有算例上本文算法 ACS_VND 的求解质量都优于文献[6]中的算法结果;而与文献[9]中算法结果相比,在 17 个算例上二者求解结果相同,而在其余 33 个算例上,本文算法 ACS_VND 的求解结果更优. 对于算例 SCA8-2、SCA8-7 和 CON8-3,本文算法的解比文献[9]中算法的解使用的车辆数少 1. 另外,对所有算例本文算法都能在 2s 内找到最好解.

表 3 算法 ACS_VND 和现有算法在 Dethloff 算例上的最好解比较

算例	<i>n</i>	Dethloff ^[6] 的结果			Tang Montané &. Galvão ^[9] 的结果			ACS_VND 的结果		
		<i>L</i>	<i>k</i>	<i>t</i> /s	<i>L</i>	<i>k</i>	<i>t</i> ^{<i>a</i>} /s	<i>L</i>	<i>k</i>	<i>t</i> ^{<i>b</i>} /s
SCA3-0	50	689.0	-	-	640.55	4	3.37	635.62	4	1.19
SCA3-1	50	765.6	-	-	697.84	4	3.25	697.84	4	1.14
SCA3-2	50	742.8	-	-	659.34	4	3.52	659.34	4	1.02
SCA3-3	50	737.2	-	-	680.04	4	3.31	680.04	4	1.19
SCA3-4	50	747.1	-	-	690.50	4	3.43	690.50	4	1.22
SCA3-5	50	784.4	-	-	659.90	4	3.67	659.90	4	1.01
SCA3-6	50	720.4	-	-	653.81	4	3.35	651.09	4	1.16
SCA3-7	50	707.9	-	-	659.17	4	3.33	659.17	4	0.81
SCA3-8	50	807.2	-	-	719.47	4	3.40	719.47	4	1.12
SCA3-9	50	764.1	-	-	681.00	4	3.41	681.00	4	1.07
SCA8-0	50	1132.9	-	-	981.47	9	4.14	961.50	9	1.08
SCA8-1	50	1150.9	-	-	1077.44	9	4.27	1049.65	9	1.17
SCA8-2	50	1100.8	-	-	1050.98	10	4.20	1044.48	9	1.20
SCA8-3	50	1115.6	-	-	983.34	9	4.17	983.34	9	1.08
SCA8-4	50	1235.4	-	-	1073.48	9	4.13	1065.49	9	0.97
SCA8-5	50	1231.6	-	-	1047.24	9	4.02	1027.08	9	1.19
SCA8-6	50	1062.5	-	-	995.59	9	3.85	971.82	9	1.40
SCA8-7	50	1217.4	-	-	1068.56	10	4.22	1052.17	9	1.32
SCA8-8	50	1231.6	-	-	1080.58	9	3.85	1071.18	9	1.19
SCA8-9	50	1185.6	-	-	1084.80	9	4.20	1060.50	9	1.09
CON3-0	50	672.4	-	-	631.39	4	3.64	616.52	4	1.71
CON3-1	50	570.6	-	-	554.47	4	3.31	554.47	4	1.53
CON3-2	50	534.8	-	-	522.86	4	3.45	518.00	4	1.23
CON3-3	50	656.9	-	-	591.19	4	3.28	591.19	4	1.24
CON3-4	50	640.2	-	-	591.12	4	3.47	588.79	4	1.40
CON3-5	50	604.7	-	-	563.70	4	3.38	563.70	4	1.27
CON3-6	50	521.3	-	-	506.19	4	3.32	501.32	4	1.20
CON3-7	50	602.8	-	-	577.68	4	3.51	576.48	4	1.18
CON3-8	50	556.2	-	-	523.05	4	3.66	523.05	4	1.37
CON3-9	50	612.8	-	-	580.05	4	3.36	578.25	4	1.24
CON8-0	50	967.3	-	-	860.48	9	4.19	857.40	9	1.65
CON8-1	50	828.7	-	-	740.85	9	3.89	740.85	9	1.39
CON8-2	50	770.2	-	-	723.32	9	3.76	712.89	9	1.65
CON8-3	50	906.7	-	-	811.23	10	4.12	829.87	9	1.32
CON8-4	50	876.8	-	-	772.25	9	3.75	772.25	9	1.31
CON8-5	50	866.9	-	-	756.91	9	3.99	754.95	9	1.45
CON8-6	50	749.1	-	-	678.92	9	4.04	678.92	9	1.42
CON8-7	50	929.8	-	-	814.50	9	4.00	811.96	9	1.10
CON8-8	50	833.1	-	-	775.59	9	3.74	767.53	9	1.60
CON8-9	50	877.3	-	-	809.00	9	4.13	809.00	9	1.60

注: *t*^{*a*}表示 Athlon 微机,2.0GHz 上的运行时间;*t*^{*b*}表示 Pentium IV 微机,2.93GHz 上的运行时间.

由于文献[11]中只给出算法求解的平均解,因此,我们在表 4 中列出算法 ACS_VND 以及现

有算法求解的平均结果. 根据表 4 中各算法在 4 组算例上的平均结果,与文献[6]中算法相比,算法

ACS_VND 的求解质量分别提高了 9.80%，11.81%，6.04%，10.11%，整体提高 9.91%；与文献[9]中算法相比，算法 ACS_VND 的求解质量分别提高了 0.11%，1.50%，0.53%，0.10%，整体提高 0.66%；

与文献[11]中算法相比，算法 ACS_VND 的求解质量分别提高了 1.64%，0.67%，1.29%，0.37%，整体提高 0.92%。另外，本文算法和文献[11]中算法平均使用的车辆数更少。

表 4 算法 ACS_VND 和现有算法在 Dethloff 算例上分组平均结果的比较												
算例	Dethloff ^[6] 的结果			Tang Montané & Galvão ^[9] 的结果			Bianchessi ^[11] 的结果			ACS_VND 的结果		
	<i>L</i>	<i>k</i>	<i>t</i> ^{<i>a</i>} /s	<i>L</i>	<i>k</i>	<i>t</i> ^{<i>a</i>} /s	<i>L</i>	<i>k</i>	<i>t</i> ^{<i>b</i>} /s	<i>L</i>	<i>k</i>	<i>t</i> ^{<i>c</i>} /s
SCA3	746.57	-	-	674.16	4	3.4	684.6	4	59.47	673.4	4	1.1
SCA8	1166.43	-	-	1044.35	9.2	4.11	1035.7	9	17.05	1028.72	9	1.17
CON3	597.27	-	-	564.17	4	3.44	568.5	4	59.36	561.18	4	1.34
CON8	860.59	-	-	774.31	9.1	3.96	776.4	9	15.14	773.56	9	1.45
Avg.	842.72	-	-	764.25	6.58	3.73	766.3	6.5	37.76	759.22	6.5	1.25

注：*t^a*表示 Athlon 微机，2.0GHz 上的运行时间；*t^b*表示微机，1.6GHz 上的运行时间；*t^c*表示 Pentium IV 微机，2.93GHz 上的运行时间。

Salhi 和 Nagy 算例有两种类型的数据：实数型需求和整数型需求。其中实数型需求是根据上文所述方法产生，而整数型需求则是通过对实数型需求

四舍五入得到。除文献[9]其余现有算法均使用实数型需求，各算法求得最好解分别在表 5 和表 6 中给出。通过表 5 可以看出，算法 ACS_VND 求得的最

表 5 算法 ACS_VND 和现有算法在 Salhi 和 Nagy 算例上的最好解比较 (实数型需求)													
算例	n	Salhi & Nagy ^[7] 的结果			Crispim & Brandão ^[8] 的结果			Chen & Wu ^[10] 的结果			ACS_VND 的结果		
		L	k	t^a/s	L	k	t^b/s	L	k	t^c/s	L	k	t^d
CMT1X	50	601	6	1.5	477	3	11.2	478.59	3	7.74	466.77	3	1.27
CMT1Y	50	603	5	1.5	485	3	9.1	480.78	3	7.81	466.77	3	1.09
CMT2X	75	873	10	1.2	710	6	24.3	688.51	6	24.86	693.50	6	5.11
CMT2Y	75	924	12	1.2	715	6	26.4	679.44	6	12.02	666.75	6	5.46
CMT3X	100	923	10	1.6	744	5	37.1	744.77	5	94.06	715.51	5	11.77
CMT3Y	100	923	10	1.7	742	5	33.5	723.88	5	120.66	724.98	5	12.55
CMT12X	100	820	10	4.9	731	5	37.1	678.46	6	46.83	669.10	5	11.77
CMT12Y	100	873	11	4.8	860	5	33.7	676.23	6	56.35	662.41	5	11.65
CMT11X	120	1500	11	3.4	944	4	32.4	858.57	4	321.08	842.58	4	26.66
CMT11Y	120	1500	11	3.5	1035	4	29.6	859.77	5	230.72	843.28	4	30.27
CMT4X	150	1178	15	12.3	915	7	58.1	887.00	7	501.95	843.24	7	48.95
CMT4Y	150	1178	15	4.3	996	7	47.6	852.35	7	406.32	861.40	7	47.00
CMT5X	199	1509	19	12.3	1136	10	89.4	1089.22	10	1055.83	1074.88	10	102.71
CMT5Y	199	1477	19	12.0	1129	10	77.1	1084.27	10	771.71	1089.88	10	102.83
Avg.		1063	11.7	4.73	829.93	5.7	39.04	770.13	5.9	261.28	758.65	5.7	29.94

注：*t^a*表示 VAX 4000-500 上的运行时间；*t^b*表示 Pentium II 微机，350MHz 上的运行时间；*t^c*表示 Pentium IV 微机，1.6GHz 上的运行时间；*t^d*表示 Pentium IV 微机，2.93GHz 上的运行时间。

表 6 算法 ACS_VND 和现有算法在 Salhi 和 Nagy 算例上的最好解比较(整数型需求)							
算例	<i>n</i>	Tang Montané & Galvão ^[9] 的结果			ACS_VND 的结果		
		<i>L</i>	<i>k</i>	<i>t</i> ^{<i>a</i>} /s	<i>L</i>	<i>k</i>	<i>t</i> ^{<i>c</i>} /s
CMT1X	50	472	3	3.73	470.67	3	1.31
CMT1Y	50	470	3	4.37	470.67	3	1.13
CMT2X	75	695	7	6.91	684.29	6	5.71
CMT2Y	75	700	7	7.61	664.54	6	4.44
CMT3X	100	721	5	11.04	715.14	5	14.33
CMT3Y	100	719	5	12.01	724.38	5	12.55
CMT12X	100	880	6	12.23	667.29	5	14.17
CMT12Y	100	878	6	12.80	666.93	5	10.70
CMT11X	120	900	4	18.17	839.05	4	27.91
CMT11Y	120	910	5	18.04	834.52	4	32.43
CMT4X	150	880	7	24.60	841.29	7	55.89
CMT4Y	150	878	7	29.07	860.32	7	49.57
CMT5X	199	1098	11	51.50	1077.22	10	114.61
CMT5Y	199	1083	10	56.21	1083.63	10	117.36
Avg.		806.0	6.1	19.16	732.02	5.7	33.01

注：*t^a*表示 Athlon 微机，2.0GHz 上的运行时间；*t^b*表示 Pentium IV 微机，2.93GHz 上的运行时间。

好解使用的车辆数(*k*)远远少于文献[7]中的算法结果，略少于文献[10]中的算法结果，与文献[8]中的算法结果相同；从路径长度来看，算法 ACS_VND 在所有算例上的解都优于文献[7-8]中的算法；除算例 CMT2X、CMT4Y 和 CMT5Y，本文算法在其余算例上的解都优于文献[10]中算法。与文献[7]中算法相比，算法 ACS_VND 的求解质量最大提高 43.83%，最小提高 18.40%；与文献[8]中的算法相比，算法 ACS_VND 的求解质量最大提高 22.98%，最小提高 2.14%；与文献[9]中算法相比，算法 ACS_VND 的求解质量最大提高 4.93%，最小提高 -1.06%。从平均结果来看，算法 ACS_VND 的解比文献[7]、[8]和[10]中算法的解分别提高 28.63%、8.59%和 1.49%。根据表 6 中的数据，与文献[9]中算法相比，算法 ACS_VND 在算例

CMT2X、CMT2Y、CMT12X、CMT12Y、CMT11Y 和 CMT5X 上都能找到车辆数比前者少 1 的解,其余算例则相同;从解的路径长度看,算法 ACS_VND 最大提高 24.17%,最小提高-0.75%,平均提高 9.18%.

4.3 算法运行时间的比较

由于各算法运行环境不同,因此不能直接比较算法的 CPU 时间. 根据 Dongarra 的 Linpack 基准测试^[17],可以通过 Mflops(Million Floationg Point/Second,每秒百万个浮点操作)标准得到不同微机的大致相对速度. 表 7 中给出了文献[7-9]中各微机的机型主频、浮点运算速度/Mflops 以及转换为 VAX4000-500 上 CPU 时间的转换因子(factor)(由于文献[11]中未能给出具体的微机型号,故此处略去). 从表 5 和表 7 可以看出,算法 ACS 的运行时间远远多于文献[7]中算法,这是因为,前者是迭代搜索算法,而后者是构造型方法,不需要多次迭代. 与文献[8-9]中算法相比,由于算法 ACS_VND 使用的机器运行速度较快,因此,算法 ACS_VND 运行时间较长. 即使考虑运行环境性能的差异,算法 ACS_VND 运行时间仍然比文献[10]中算法运行时间短. 由表 3、表 5 和表 6 可以看出,对于较小规模问题,算法 ACS_VND 能够在很短时间内获得较好解,随着问题规模的增大,虽然算法 ACS_VND 所需 CPU 时间增长较快,但是对于规模较大的问题如 $n=199$,仍能在 2min 之内获得较好解.

表 7 各算法运行环境性能比较

	CPU	运算速度/ Mflops	factor
Salhi & Nagy ^[7]	VAX 4000-500	5.7	1
Crispim & Brandão ^[8]	Pentium II, 350MHz	69~74	10
Tang Montané & Galvão ^[9]	Athlon, 2. 0GHz	832~1000	170
Chen & Wu ^[10]	Pentium IV, 1. 6GHz	<1190	<209
ACS_VND	Pentium IV, 2. 93GHz	1317~1414	240

5 结 论

结合不同元启发式方法的优点和策略,设计更有效的混合启发式算法是组合优化问题研究领域的一个热点. 结合蚁群系统和变邻域下降搜索,本文提出一种混合启发式算法 ACS_VND,用于求解卸装一体化车辆路径问题. 利用蚁群系统的搜索多样性与变邻域下降搜索较强的局部寻优能力,提高求解质量,加速算法的收敛. 通过实验验证了混合算法

ACS_VND 性能优于单一算法 ACS 和 VND. 另外,对 55 个 benchmark 算例的求解结果表明,算法 ACS_VND 能在较短时间内获得 52 个算例的已知最好解,并且更新了 44 个算例的已知最好解,表明本文算法的表现优于现有算法.

参 考 文 献

[1] Dantzig G B, Ramser J H. The truck dispatching problem. *Management Science*, 1959, 6(1): 80-91

[2] Min H. The multiple vehicle routing problem with simultaneous delivery and pickup points. *Transportation Research A*, 1989, 23A(5): 377-386

[3] Mosheiov G. The traveling salesman problem with pick-up and delivery. *European Journal of Operational Research*, 1994, 79(2): 299-310

[4] Privé J, Renaud J, Boctor F, Laporte G. Solving a vehicle routing problem arising in soft drink distribution. *Journal of the Operational Research Society*, 2006, 57(9): 1045-1052

[5] Wasner M, Zapfel G. An integrated multi-depot hub-location vehicle routing model for network planning of parcel service. *International Journal of Production Economics*, 2004, 90(3): 403-419

[6] Dethloff J. Vehicle routing and reverse logistics; The vehicle routing problem with simultaneous delivery and pick-up. *OR Spektrum*, 2001, 23(1): 79-96

[7] Salhi S, Nagy G. A cluster insertion heuristic for single and multiple depot vehicle routing problem with backhauling. *Journal of the Operational Research Society*, 1999, 50(10): 1034-1042

[8] Crispim J, Brandão J. Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls. *Journal of the Operational Research Society*, 2005, 56(11): 1296-1302

[9] Tang Montané F A, Galvão R D. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, 2006, 33(3): 595-619

[10] Chen J F, Wu T H. Vehicle routing problem with simultaneous deliveries and pickups. *Journal of the Operational Research Society*, 2006, 57(5): 579-587

[11] Bianchessi N, Righini G. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers & Operations Research*, 2007, 34(2): 578-594

[12] Dorigo M, Stützle T. *Ant Colony Optimization*. Cambridge, Massachusetts, London, England: The MIT Press, 2004

[13] Hansen P, Mladenović N. An introduction to variable neighborhood search//Voß S, Martello S, Osman S, Roucairol C eds. *Metaheuristic: Advanced and Trends in Local Search Paradigms for Optimization*. Boston: Kluwer Academic Publishers, 1999: 433-458

[14] Hansen P, Mladenović N. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 2001, 130(3): 449-467

[15] Blum C. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 2003, 35(3): 268-308

[16] Christofides N, Mingozi A, Toth P. The vehicle routing problem//Christofides N, Mingozi P, Toth P, Sandi C eds. *Combinatorial Optimization*. Wiley: Chichester, 1979: 315-338

[17] Dongarra J J. Performance of various computers using standard linear equations software. Computer Science Department, University of Tennessee, Knoxville, TN: Technical Report CS-89-85, 2007



CHEN Ping, born in 1981, Ph. D. candidate. Her research interests include optimization methods for vehicle routing problem, metaheuristic and combinatorial optimization.

HUANG Hou-Kuan, born in 1940, professor, Ph. D. supervisor. His research interests include artificial intelligence, machine learning, data warehousing, data mining, decision support system, and multi-agent system, etc.

DONG Xing-Ye, born in 1974, Ph. D. candidate. His research interests include planning and scheduling algorithms and application of optimization methods in combinatorial problems.

Background

Distribution of goods is of paramount importance in the logistics and supply chain management. Since the transportation cost occupies a great proportion in the total logistics cost, dispatching and routing the vehicles in a rational way may considerably cut off the cost of the logistics companies. Furthermore, the energy consumption may be also reduced, which is good for the environmental protection. In this paper, a variant of vehicle routing problem, i. e. , vehicle routing problem with simultaneous delivery and pickup is discussed. It has a strong background in the reverse logistics and has attracted researchers' interests recently.

The vehicle routing problem is NP-hard, heuristic and

metaheuristic methods, which can find quite good solutions in a very short time, are practical for solving this problem. Recently, it has become evident that the concentration on a sole metaheuristic is rather restrictive. Thus the hybrid heuristic methods, which are to combine a metaheuristic with other optimization techniques, can behave more efficiently and flexibly when dealing with real-world and large-scale problems. In this paper, a new hybrid heuristic algorithm is proposed, which can find 52 best known solutions and update 44 best known solutions out of 55 benchmark problems. This work is supported by the National Basic Research Program (973 Program) of China under grant No. 2006CB705500.