

Creating Very Large Scale Neighborhoods out of Smaller Ones by Compounding Moves: A Study on the Vehicle Routing Problem

Özlem Ergun • James B. Orlin • Abran Steele-Feldman

*Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA
30332-0205, USA*

*Operations Research Center, E40-147 Massachusetts Institute of Technology, Cambridge, MA
02139, USA*

oergun@isye.gatech.edu • jorlin@mit.edu • abranskee@yahoo.com

Neighborhood search algorithms are a wide class of improvement algorithms where at each iteration an improving solution is found by searching the “neighborhood” of the current solution. This paper discusses neighborhood search algorithms where the size of the neighborhood is “very large” with respect to the size of the input data. For large problem instances, it is impractical to search these neighborhoods explicitly, and one must either search a small portion of the neighborhood or else develop efficient algorithms for searching the neighborhood implicitly. We concentrate on a very large scale neighborhood (VLSN) search technique based on compounding independent moves (CIM) such as 2-opt, swaps, and insertions. We demonstrate that the search for an improving neighbor can be done by finding a negative cost path on an auxiliary graph. In this paper we study CIM algorithms for the vehicle routing problem with capacity and distance constraints. We present results of the computational study which indicates that the CIM algorithms for the capacitated vehicle routing problem are competitive with the current state of the art heuristics.

(Very-large scale neighborhood search; Vehicle routing problem; Heuristics)

1 Introduction

In this paper, we consider very large scale neighborhood (VLSN) search applied to the traveling salesman problem and the vehicle routing problem. Given a solution S to an instance of an optimization problem, a *neighborhood function* defines the neighborhood, $N(S)$, of S . Often a neighborhood function is expressed in terms of elementary transformations of S , called “moves”, that transform S to another solution. 2-opt, swap, and insertion moves are common examples of moves. A local improvement algorithm starts with a current solution S and then searches $N(S)$

for a better solution. If a better solution exists, then one such solution is selected to replace the current solution and the algorithm repeats. If there are no better solutions in $N(S)$ the algorithm terminates with the current solution as the local optimum with respect to N . A critical issue in the design of a neighborhood search approach is the choice of the neighborhood structure, that is, the manner in which the neighborhood is defined. This choice largely determines whether the neighborhood search will yield solutions that are highly accurate or whether they will develop solutions with very poor local optima.

In this paper, we study a class of neighborhood search algorithms where the size of the neighborhood is “very large”, often exponential, with respect to the size of the input data along with efficient algorithms for searching these neighborhoods. In particular, we develop a very large scale neighborhood, which we name the compounded independent moves (CIM) neighborhood. The research extends and generalizes earlier research by Potts and van de Velde [22], Congram [8], and Congram et al. [9]. We concentrate on the CIM neighborhood obtained by compounding independent 2-opt, swap, and insertion moves. We show that for the traveling salesman problem, these CIM neighborhoods can be searched in polynomial time for an improving neighbor by finding a negative cost path on an auxiliary graph. Furthermore, we extend that the CIM neighborhood using network flows based search algorithms and additional ideas to the vehicle routing problem with side constraints. We present a computational study on the capacity and distance restricted vehicle routing problem. The compounded independent move neighborhoods applied to a variety of routing and scheduling problems are covered in more detail in [12]. For surveys of VLSN search see [2] and [10].

2 CIM Neighborhoods for the Traveling Salesman Problem

An exponentially sized neighborhood for the traveling salesman Problem (TSP) can be obtained by simultaneously combining a set of independent moves. Approaches that rely on combining elementary moves have been explored and surveyed for the TSP by Deineko and Woeginger [10], and applied to various sequencing problems including the TSP and some scheduling problems by Congram et al. [9], and Potts and van de Velde [22]. Related TSP neighborhoods based on sequential adding and deleting of edges are studied in [13], [17], and [23].

In this section, we explain how to construct and efficiently search the “compounded independent 2-opt neighborhood” for the symmetric TSP. Then we extend these results for the compounded independent insertions and swaps neighborhoods. Subsequently, we enlarge these neighborhoods

by weakening the independence notion for compounding moves. All of these neighborhoods have an exponential size, and we develop formulae for the growth in these neighborhoods.

Our major contribution in this section is to use an auxiliary graph for searching the compounded independent 2-opt neighborhood. Potts and van de Velde [22] solve the search problem directly using dynamic programming. By our construction of the improvement graph, we can solve the search problem as a shortest path problem, and we can solve constrained search problems as constrained shortest path problems.

2.1 Compounded Independent 2-opt Neighborhood for the TSP

In this sub-section, we consider 2-opt moves, also called 2-exchanges. A tour T' is obtained from a tour T via a 2-opt move if T' can be obtained by adding two edges to T and deleting two edges. We call T' a 2-opt neighbor of T . Let T be a TSP tour on n cities, and let $N(T)$ be the set of 2-opt neighbors of T . We present T as a sequence of cities $T = i_1, i_2, \dots, i_n, i_1$. We let $loc_T(j)$ denote the city in location j . In this case, $loc_T(j) = i_j$. In general, $T =: (loc_T(1), \dots, loc_T(n), loc_T(1))$. This permits n different representations describing the same tour. If we want the description to be unique, we will specify that $loc_T(1) = 1$. For tour T , we let $rank_T(j)$ denote the position of city j in the tour. Hence, $loc_T(i) = j$ if and only if $rank_T(j) = i$. We let $f_T(i, j)$ be the cost of traveling from city $loc_T(i)$ to city $loc_T(j)$.

For the simplicity of the presentation we will drop the identifier T from the $loc()$, $rank()$, and $f(,)$ whenever associating these functions with a specific tour is clear from the context.

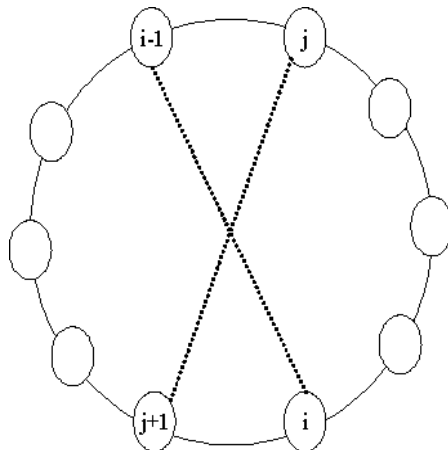


Figure 1

We let $2\text{-opt-move}(i, j)$ denote the move that transforms T into T' by deleting two arcs, $(loc(i -$

1), $loc(i)$, and $(loc(j), loc(j+1))$, in T and adding two new arcs, $(loc(i-1), loc(j))$ and $(loc(i), loc(j+1))$ that are not in T . This exchange also causes the order of the cities in $loc(i+1)$ to $loc(j-1)$ to be inverted. In $2\text{-opt-move}(i, j)$ we assume $i < j$. A single 2-opt move is illustrated in Figure 1, where dashed lines indicate the edges removed from the tour.

We say that a move is *restricted to the interval* $[i, j]$ if the following is true: If tour S is obtained from T by applying the move, then $loc_S(k) = loc_T(k)$ for $k < i$ and for $k > j$. $2\text{-opt-move}(i, j)$ and $2\text{-opt-move}(k, l)$ are *independent* if either $j + 1 < k$ or $l + 1 < i$. We say that a collection S of moves is independent if every two moves in S is independent.

The *compounded independent 2-opt neighborhood* for tour T is the set of all tours T' which can be obtained by applying a collection of pairwise independent 2-opt moves to T . This neighborhood is illustrated in Figure 2.

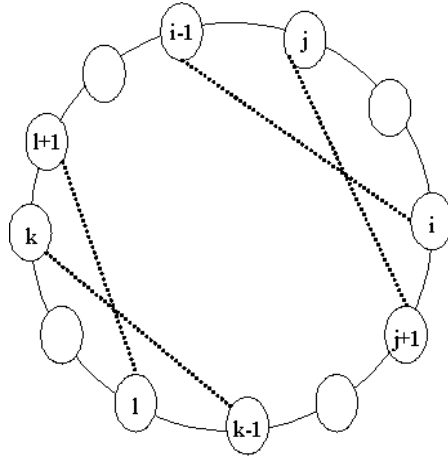


Figure 2

Let $cost(T)$ be the cost of visiting the cities in the order they appear in tour T . Then $\bar{c}^1(i, j)$, the incremental change in tour T 's cost after $2\text{-opt-move}(i, j)$ is applied, is equal to $cost(T') - cost(T)$. Now we can state the following two propositions:

Proposition 1 *Suppose that S is a set of independent 2-opt moves. Then the moves in set S can be carried out sequentially in any order, the same tour T' will be obtained from tour T independent of the order in which the moves are carried out, and the cost of the moves in S is $c^1(S) = cost(T') - cost(T) = \sum_{(i,j) \in S} \bar{c}^1(i, j)$.*

Proposition 2 *Suppose that S is an ordered collection of moves such that for any two consecutive moves $move(i, j)$ and $move(k, l)$, $j + 1 < k$. Then the collection S is independent, and $c^1(S) =$*

$$\sum_{(i,j) \in S} \bar{c}^1(i,j).$$

The compounded independent 2-opt neighborhood can be searched by finding a shortest path on a related network structure that we call the *improvement graph* as per [27].

Given a TSP tour $T = (loc(1), loc(2), \dots, loc(n), loc(1))$ the improvement graph $G(V, E)$ associated with T is defined by the node set $V = \{1, 2, \dots, n, n+1\}$ where each node i represents the city $loc(i)$ for $1 \leq i \leq n$, and node $n+1$ represents the fact that the loop is completed by returning back to city $loc(1)$. An arc (i, j) in the improvement graph represents $2\text{-opt-move}(i+1, j-1)$, a move restricted to the interval $[i+1, j+1]$, that is involving only cities $loc(i+1), loc(i+2), \dots, loc(j-1)$. The arc set E consists of all arcs (i, j) such that $1 \leq i \leq n$ and $j = i+1$ or $i+3 \leq j$. Note that the arc set E only consists of forward arcs, which insures that G is acyclic and which in turn assures by Proposition 2 that the compounded moves that are identified with the search algorithm are independent. Furthermore, we will associate an independent set of 2-opt moves with a path from 1 to $n+1$. There are no arcs of the form $(i, i+2)$ since there is no 2-opt move involving only city $loc(i+1)$. For convenience, we let $f(n, n+1) = f(n, 1)$. The cost structure on the arcs in E is constructed as follows:

- $c^1(i, i+1) = 0$. Arc $(i, i+1)$ for $1 \leq i \leq n$ will be in the path when cities $loc(i)$ and $loc(i+1)$ are not involved in any moves.
- Arc (i, j) for $1 \leq i < n-1$ and $i+2 < j \leq n+1$ represents a $2\text{-opt-move}(i+1, j-1)$.

$$c(i, j)^1 = -(f(i, i+1) + f(j-1, j)) + (f(i, j-1) + f(i+1, j)).$$

Note that when we are concerned with the asymmetric TSP the cost term should also include the impact of the reversal of the subtour $(loc(i+1), \dots, loc(j-1))$. Figure 3 illustrates a partial improvement graph for compounded 2-opts.

Theorem 3 *There is a 1 to 1 cost-preserving correspondence between paths from node 1 to node $n+1$ in the compounded independent 2-opt improvement graph $G(V, E)$ and sets of independent 2-opt moves.*

Proof We first suppose P is a path from 1 to $n+1$ in G . let P' be the set of edges obtained by deleting all edges of the form $(i, i+1)$ in P . Then the edges of P' correspond to an ordered collection of 2-opt moves such that for any two consecutive moves $move(i, j)$ and $move(k, l)$, $j+1 < k$. By

Property 2, this collection of moves is independent, and its cost is equal to $\sum_{(i+1,j-1) \in S} \bar{c}^1(i+1, j-1) = \sum_{(i,j) \in S} c^1(i, j)$ which is also the cost of path P .

We next suppose we are given a set S of ordered independent 2-opt moves. Let P' be the corresponding edges in E , ordered in increasing order of index. We extend P' to a path P by adding edges of the form $(i, i+1)$. Then the cost of P is equal to $c(S) = \sum_{(i,j) \in S} c^1(i, j) = \sum_{(i+1,j-1) \in S} \bar{c}^1(i+1, j-1)$. •

Finding any negative cost path from node 1 to node $n+1$ on the improvement graph $G(V, E)$ corresponds to finding a combination of independent 2-opt moves that transforms the current tour T into a new tour $T' \in N(T)$ with a lower cost. If the length of the shortest path from 1 to $n+1$ has a cost of 0, then there are no improving neighbors of T , and this is a locally optimal tour under the compounded independent 2-opt neighborhood. Searching the compounded independent 2-opt neighborhood takes $O(n^2)$ times if the arc costs on E either are given or can be calculated in constant time per arc.



Figure 3. Let tour $T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1)$. After applying the CI 2-opt moves implied by the path $1 - 2 - 3 - 6 - 11 - 12$ tour $T = (1, 2, 3, 5, 4, 6, 10, 9, 8, 7, 11, 1)$ is obtained.

Although the compounded independent move neighborhood is exponentially large, it shares a key feature with the much smaller 2-opt neighborhood.

Theorem 4 *The set of local optimal solutions with respect to the compounded independent 2-opt neighborhood is the same as the set of local optimal solutions for the 2-opt neighborhood.*

Proof See [22]. •

2.2 Compounded Independent Insertion and Swap Neighborhoods for the TSP

The compounded independent 2-opt neighborhood can be directly extended to the move types swaps and insertions as well as other move types. The swap neighborhood of a given tour $T = (loc_T(1), \dots, loc_T(n), loc(1))$ consists of all tours T' which can be obtained by switching the locations of two

cities $loc_T(i)$ and $loc_T(j)$ in T . Thus $loc_{T'}(i) = loc_T(j)$, $loc_{T'}(j) = loc_T(i)$, and $loc_{T'}(k) = loc_T(k)$ for all $k \neq i$ or j . We refer to the swap as $swap-move(i, j)$ and we assume $i < j$. We say that $swap-move(i, j)$ is *independent* of $swap-move(k, l)$ if $j + 1 < k$ or $l + 1 < i$. The compounded independent swaps neighborhood for tour T is the set of all tours T' which can be obtained by applying a collection of independent swap moves to T . Propositions 1 and 2 of Section 2.1 directly extend for a collection of independent swap moves. Moreover, the improvement graph for swaps is identical to the one described in Section 2.1 with the exception of the arc costs. The arc costs $c^2(i, j)$ in this improvement graph is calculated as follows:

- $c^2(i, i+1) = 0$. Arc $(i, i+1)$ for $1 \leq i \leq n$ will be in the path when cities $loc(i)$ and $loc(i+1)$ are not involved in any moves.
- Arc (i, j) for $1 \leq i < n-1$ and $i+2 < j \leq n+1$ represents a $swap-move(i+1, j-1)$. Thus links $(loc(i), loc(i+1))$, $(loc(i+1), loc(i+2))$, $(loc(j-2), loc(j-1))$, and $(loc(j-1), loc(j))$ in T are broken and the links $(loc(i), loc(j-1))$, $(loc(j-1), loc(i+2))$, $(loc(j-2), loc(i+1))$ and $(loc(i+1), loc(j))$ are established. The cost on these arcs is:

$$c^2(i, j) = -(f(i, i+1) + f(i+1, i+2) + f(j-2, j-1) + f(j-1, j)) \\ + (f(i, j-1) + f(j-1, i+2) + f(j-2, i+1) + f(i+1, j)).$$

The insertion neighborhood of a given tour $T = (loc_T(1), \dots, loc_T(n), loc(1))$ consists of all tours T' which can be obtained by ejecting a city from its current location and inserting it into a different location. Suppose $i < j$. we let $insertion-move(i, j)$ be the move such that the node $loc_T(i)$ is inserted immediately before the node $loc_T(j)$. The resulting tour T' has $loc_{T'}(k) = loc_T(k)$ for all $k < i$ and $k \geq j$. Also, $loc_{T'}(k) = loc_T(k+1)$ for $k = i$ to $j-2$, and $loc_{T'}(j-1) = loc_T(i)$. Independence of insertion moves is defined the same as for 2-opt moves and swap moves, the compounded independent insertions neighborhoods we use in this paper is the set of all tours T' which can be obtained by applying a collection of independent forward insertion moves where a city $loc(i)$ is ejected from its current location and is inserted between two cities $loc(j)$ and $loc(j+1)$ for all $i < j$. Propositions 1 and 2 of Section 2.1 directly extend for a collection of independent insertion moves. Moreover, the improvement graph for insertion moves is identical to the one described in Section 2.1 with the exception of the arc costs. The arc costs $c^3(i, j)$ in this improvement graph is calculated as follows:

- $c^3(i, i+1) = 0$. Arc $(i, i+1)$ for $1 \leq i \leq n$ will be in the path when cities $loc(i)$ and $loc(i+1)$ are not involved in any moves.
- Arc (i, j) for $1 \leq i < n-1$ and $i+2 < j \leq n+1$ represents an *insertion-move* $(i+1, j-1)$. That is the links $(loc(i), loc(i+1))$, $(loc(i+1), loc(i+2))$, $(loc(j-1), loc(j))$ in T are broken and the links $(loc(i), loc(i+2))$, $(loc(j-1), loc(i+1))$ and $(loc(i+1), loc(j))$ are established. The cost on these arcs is:

$$\begin{aligned}
c(i, j)^3 = & -(f(i, i+1) + f(i+1, i+2) + f(j-1, j)) \\
& + (f(i, i+2) + f(j-1, i+1) + f(i+1, j)).
\end{aligned}$$

For $i < j$, we could also consider a *reverse-insertion move* in which the node $loc_T(j)$ is inserted directly before the node $loc_T(i)$. We can compute the cost $c^4(i, j)$ of *reverse-insertion-move* $(i+1, j-1)$ in a similar manner to computing the cost $c^3(i, j)$.

Given the above structure of the improvement graphs for the compounded independent swap and insertion neighborhoods, it is easy to see that the Theorems 3 and 4 can directly be extended for these neighborhoods as well. Furthermore, search for an improving neighbor can be done in $O(n^2)$ time by finding a shortest path on the appropriate improvement graph.

Moreover, one can accommodate the compounding of different types of moves in one neighborhood. For example, a larger neighborhood obtained by compounding independent 2-opt, swap, insertion, and reverse-insertion moves can be modeled and searched by constructing an identical improvement graph and letting the arc cost $c(i, j) = \min(c^1(i, j), c^2(i, j), c^3(i, j), c^4(i, j))$.

2.3 Compounded Weakly Independent Moves Neighborhoods

By weakening the independence requirement for compounding moves one can obtain larger sized neighborhoods. We say that *2-opt-move* (i, j) is *weakly independent* of *2-opt-move* (k, l) if $j+1 \leq k$ or $l+1 \leq i$. We define weak independence for swap, insertion, and reverse-insertion moves in the same manner. Then the compounded weakly independent 2-opt (swap, insertion, reverse-insertion) neighborhood for tour T is the set of all tours T' which can be obtained by applying a collection of weakly independent 2-opt (swap, insertion, reverse-insertion) moves to T .

In general, the size of the compounded weakly independent moves neighborhood is asymptotically larger than the size of the compounded independent moves neighborhood. We give asymptotic bounds for the exponential growth of these neighborhoods in Section 2.4. At the same time, the

increased size of these neighborhoods comes at a cost in search time. The compounded weakly independent 2-opt moves neighborhood apparently takes n times as long to search as the compounded independent 2-opt moves neighborhood. (At least, we could not find a faster approach.) The same increase in running time applies for the swap-based neighborhoods. However, we will show below that the compounded weakly independent insertion neighborhood takes only a constant times longer to search as the compounded independent insertion neighborhood.

Let $G = (V, E)$ be a graph with arc costs $c(e_j)$ for $e_j \in E$. A *turn penalty* (see [19]) is a function $g : E \times E \rightarrow \mathfrak{R}$. If $e \in E$ and $e' \in E$, then $g(e, e')$ is the (additional) cost of having arc e' directly follow arc e in a path. Then the cost of a path consisting of arcs e_1, \dots, e_k is $\sum_{j=1}^k c(e_j) + \sum_{j=1}^{k-1} g(e_j, e_{j+1})$.

The compounded weakly independent 2-opt, swap, insertion, and reverse-insertion neighborhoods can be searched by finding shortest paths with turn penalties on improvement graphs $G(V, E)$ constructed similarly to the ones described in the previous sections. As before T is the current tour. In these improvement graphs the node set is $V = \{1, 2, \dots, n, n + 1\}$, where each node i represents the city $loc_T(i)$ for $1 \leq i \leq n$ and node $n + 1$ represents the fact that the loop is completed by returning back to city $loc_T(1)$. An arc (i, j) in the improvement graph represents *2-opt-move* $(i, j - 1)$, which is a move that involves nodes $loc_T(i), loc_T(i + 1), \dots, loc_T(j - 1)$. The arc set E consists of all arcs (i, j) such that $1 < i \leq n$ and $i + 1 \leq j$. The cost on the improvement graph arcs depend not only on the move they represent but also on the move represented by the previous arc. The turn penalties for the arc $(j + 1, l)$ preceded by an arc $(i, j + 1)$ in the CWI 2-opt, swap, insertion, and reverse-insertion neighborhood improvement graphs can be calculated as follows:

- For arc $(j + 1, j + 2)$ the turn penalties for the 2-opt, swap, insertion, and reverse-insertion moves are equal to 0 since these arcs imply that node $j + 1$ is not part of a move.
- For arcs $(j + 1, l)$ where $l > j + 2$, for the CWI 2-opt neighborhood:

$$f(j, j + 1) + f(i, l - 1) - f(i, j + 1) - f(j, l - 1)$$

for the CWI swap neighborhood:

$$f(j, j + 1) + f(i, l - 1) - f(i, j + 1) - f(j, l - 1)$$

for the CWI insertion neighborhood:

$$f(j, j + 1) + f(i, j + 2) - f(i, j + 1) - f(j, j + 2)$$

for the CWI reverse-insertion neighborhood:

$$f(j, j + 1) + f(j - 1, l - 1) - f(j - 1, j + 1) - f(j, l - 1).$$

The following theorem is a direct extension of Theorem 1.

Theorem 5 *There is a 1 to 1 correspondence between paths from node 1 to node $n+1$ in the compounded weakly independent 2-opt (swap, insertion, reverse-insertion) improvement graph and sets of weakly independent 2-opt (swap, insertion, reverse-insertion) moves. Moreover, the cost of the weakly independent 2-opt (swap, insertion, reverse-insertion) moves is the same as the cost of the path taking into account turn penalties.*

In general, these shortest paths with turn penalties can be found in $O(n^3)$ time. However the following theorem shows that the quality of the local optimal solutions obtained by CWI moves is at least as good as the ones obtained by CI moves.

Theorem 6 *For $n \geq 5$, the set of local optimal solutions for the compounded weakly independent 2-opt (swap, insertion, reverse-insertion) neighborhood strictly contains from the set of local optimal solutions for the 2-opt (swap, insertion, reverse-insertion) neighborhood search.*

Proof Every single 2-opt (swap, insertion, reverse-insertion) move is also a feasible move for the CWI 2-opt (swap, insertion, reverse-insertion) neighborhood. Hence the feasible solutions in the compounded weakly independent 2-opt (swap, insertion, reverse-insertion) neighborhood contains the set of local optimal solutions for the 2-opt (swap, insertion, reverse-insertion) neighborhood. Now consider the tour $T = (1, 2, 3, 4, 5, 6, 7, 8, 1)$ on eight cities. Let the cost on arc between cities i and j be equal to 2 for all i and j except for the city pairs $(2, 7)$, $(1, 6)$, and $(3, 8)$. Assume that the cost on arc $(2, 7)$ is equal to 1 and the costs on arcs $(1, 6)$ and $(3, 8)$ are equal to 4. We claim that tour T is a locally optimal tour under the 2-opt neighborhood. Any 2-opt move deletes two edges in T with a total cost of 4. For the 2-opt move to be profitable the total cost of the two edges added must be less than 4. This can only be achieved if edge $(2, 7)$ is one of the two edges added to the tour. However, if edge $(2, 7)$ is one of the edges added then either edge $(1, 6)$ or edge $(3, 8)$ must also be added. The total cost of the edges added in both cases is equal to 5 which makes any 2-opt move involving the edge $(2, 7)$ non-profitable. Hence there does not exist a profitable 2-opt move for tour T .

On the other hand, consider $2\text{-opt-move}(2, 4)$ and $2\text{-opt-move}(5, 7)$. Applying these two moves on T simultaneously, we obtain tour $T' = (1, 4, 3, 2, 7, 6, 5, 8, 1)$. By definition T' is in the CWI 2-opt neighborhood of T . Furthermore, since the cost of T is equal to 16, and the cost of T' is equal to 15, proving that T is not a local optima under the CWI 2-opt moves. Examples can be constructed similarly where a locally optimal tour under swap, insertion, and reverse-insertion neighborhoods is not locally optimal under the CWI swap, insertion, and reverse-insertion neighborhoods, respectively. •

The turn penalties for the CWI insertion neighborhood have a simpler form, and this neighborhood can be searched in $O(n^2)$ on a specially constructed improvement graph as follows. Let $G(V^*, E^*)$ be an improvement graph for the compounded weakly insertions neighborhood. Let $V^* = V \cup V'$ where $V = \{1, 2, \dots, n, n+1\}$ represents the n nodes of the current tour T and $V' = \{2', 3', \dots, n-2', n-1'\}$ duplicate the nodes 2 through $n-1$.

Similar to before, we will construct G^* so that there is a one-to-one correspondence between the paths in G^* from 1 to $n+1$ and weakly independent sets of insertion moves. We describe the correspondence as we describe the construction of E^* . We let P denote a path from 1 to $n+1$ in G^* , and we let T denote the current tour.

- $(i, i+1) \in E^*$ with cost 0 for $1 \leq i \leq n$. If $(i, i+1) \in P$, this is interpreted as $loc_T(i)$ and $loc_T(i+1)$ are not in an insertion move.
- $(i, j) \in E^*$ for $1 \leq i < n-1$ and $i+2 < j \leq n+1$. If $(i, j) \in P$, this is interpreted as $insertion\text{-move}(i+1, j-1)$. The cost on arc (i, j) is

$$\begin{aligned} c^5(i, j) = & -(f(i, i+1) + f(i+1, i+2) + f(j-1, j)) \\ & + (f(i, i+2) + f(j-1, i+1) + f(i+1, j)). \end{aligned}$$

- $(i, j') \in E^*$ for $1 \leq i < n-3$ and $i+2 < j \leq n-1$. If $(i, j') \in P$ this is interpreted as $insertion\text{-move}(i+1, j)$ followed directly by $insertion\text{-move}(j, k)$ for some $k \geq j+1$. Thus $loc_T(i+1)$ will be inserted between $loc_T(j-1)$ and $loc_T(j+1)$. The cost on arc (i, j') is

$$\begin{aligned} c^5(i, j') = & -(f(i, i+1) + f(i+1, i+2) + f(j-1, j) + f(j, j+1)) \\ & + (f(i, i+2) + f(j-1, i+1) + f(i+1, j+1)). \end{aligned}$$

- $(i', j') \in E^*$ for $2 \leq i < n-3$ and $i+2 < j \leq n-1$. If $(i', j') \in P$ this is interpreted as inserting $loc_T(i)$ between $loc_T(j-1)$ and $(loc_T(j+1))$ simultaneously with an $insertion\text{-move}(j, k)$ for

some $k \geq j + 1$. The cost on arc (i', j') is

$$c^5(i', j') = -(f(j - 1, j) + f(j, j + 1)) + (f(j - 1, i) + f(i, j + 1)).$$

- $(i', j) \in E^*$ for $2 \leq i < n - 1$ and $i + 2 < j \leq n + 1$. If $(i', j) \in P$, this is interpreted as inserting $loc_T(i)$ between $loc_T(j - 1)$ and $loc_T(j)$. The cost on arc (i', j) is

$$c^5(i', j) = -(f(j - 1, j)) + (f(j - 1, i) + f(i, j + 1)).$$

The neighborhood is searched by finding a shortest path in $G(V^*, E^*)$ from node 1 to node $n + 1$. Figure 4 illustrates an example with a partial improvement graph for the compounded weakly independent insertions.

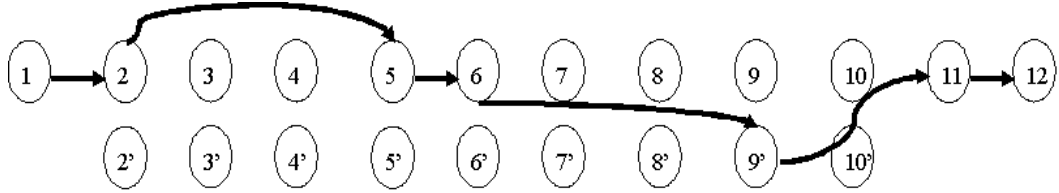


Figure 4. Let tour $T = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1)$. After applying the CWI insertion moves implied by the path $1 - 2 - 5 - 6 - 9' - 11 - 12$ tour $T = (1, 2, 4, 3, 5, 6, 8, 7, 10, 9, 11, 1)$ is obtained.

2.4 Size of the Neighborhood

The number of neighbors in the compounded independent 2-opt (swap, insertion, reverse-insertion) neighborhood is equal to the number of paths from node 1 to $n + 1$ in the improvement graph described in Section 2. We give a recursion for this number next, and solve the recursion to provide asymptotic formulae for the number of paths.

Let $g(n)$ = the number of paths from node 1 to node $n + 1$ and $g(n - 1)$ = the number of paths from node 1 to node n in the improvement graph.

Then

$$g(n) = g(n - 1) + g(n - 3) + g(n - 4) + \dots + g(1) + g(0). \quad (1)$$

This can be shown by observing that the first arc of the path goes from node 1 to one of the following nodes: 2, 4, 5, ..., $n + 1$. (Note that there are no arcs between node 1 and node 3 since the

improvement graph does not have arcs of the form $(i, i + 2)$.) Then the number of paths from node $i \in \{2, 4, 5, \dots, n + 1\}$ to node $n + 1$ is $g(n - i + 1)$.

From the above recursion it follows that:

$$g(n - 1) = g(n - 2) + g(n - 4) + g(n - 5) + \dots + g(1) + g(0). \quad (2)$$

Subtracting 2 from 1, we get an equivalent third order linear recurrence

$$g(n) = 2g(n - 1) - g(n - 2) + g(n - 3) \quad (3)$$

$$\Rightarrow g(n) - 2g(n - 1) + g(n - 2) - g(n - 3) = 0 \quad (4)$$

with the boundary conditions $g(0) = g(1) = 1$.

Using standard results in solving linear recurrence relations (see, e.g. pages 312-314 of [21]) we obtain the characteristic equation of recurrence 4, which is equal to

$$x^3 - 2x^2 + x - 1 = 0. \quad (5)$$

The largest real root of the characteristic equation 5 is between 1.7548 and 1.7549. Again by standard results in solving linear recurrences, we conclude that the largest homogeneous solution for the recurrence 6 is bounded below by $A \times 1.7548^n$ and bounded above by $A \times 1.7549^n$ for some constant A and for all sufficiently large values of n . Hence the size of the compounded independent moves neighborhood is $O(1.7549^n)$ and $\Omega(1.7548^n)$.

Furthermore, if the union of k different neighborhoods is taken as described in Section 2.2 then the size of this larger neighborhood can be calculated by solving the following recursion:

$$g(n) = 2g(n - 1) - g(n - 2) + kg(n - 3)$$

$$\Rightarrow g(n) - 2g(n - 1) + g(n - 2) - kg(n - 3) = 0$$

with the boundary conditions $g(0) = g(1) = 1$. In the case of $k = 2$, the solution to this recurrence relation is 2^n .

For the CWI 2-opt (swap, insertion, reverse-insertion) neighborhood, the recursion is as follows: $g(n) = g(n - 1) + g(n - 2) + \dots + g(1)$. This reduces to $g(n) = 2g(n - 1)$, with a root of 2. Thus the number of neighbors is $\theta(2^n)$, which is asymptotically greater than the number of neighbors for the CI 2-opt (swap, insertion, reverse-insertion) neighborhood.

3 The Vehicle Routing Problem

In this section, we consider the vehicle routing problem (VRP) and generalize the CIM heuristics of the previous section. We first describe the notation and constraints for the VRP.

We consider a vehicle routing problem with m vehicles in which cities $1, 2, \dots, n$ need to be visited. The distance from city i to city j is $d(i, j)$. The maximum distance that vehicle k is permitted to travel is D_k for $k = 1$ to m . There is a demand q_i and an additional distance penalty b_i at each city i , and a capacity C_k for vehicle k for each $k = 1$ to m . There is a single depot that is labeled as city 0. A *routing* $T = (T_1, T_2, \dots, T_m)$ is a collection of m subtours such that each city in $\{1, 2, \dots, n\}$ appears in exactly one of the subtours, and each subtour begins and ends with the depot. The VRP with distance constraints is formulated as follows:

$$\begin{aligned} \min \quad & \sum_{k=1}^m \sum_{(i,j) \in T_k} d(i, j) \\ \text{subject to} \quad & \sum_{i \in T_k} q_i \leq C_k \quad \text{for } k = 1 \text{ to } m \\ & \sum_{(i,j) \in T_k} d(i, j) + \sum_{i \in T_k} b_i \leq D_k \quad \text{for } k = 1 \text{ to } m \\ & (T_1, \dots, T_m) \text{ is a routing.} \end{aligned}$$

Any routing T satisfying the first and second constraints is called a *feasible routing*.

The VRP models the problem of determining optimal delivery or collection routes of a truck fleet through a set of locations. The VRP is NP-hard, and is very difficult to solve in practice. Problems with more than 50 cities are rarely solvable to optimality [24]. As a result of the VRP's inherent intractability, there is a large body of research on heuristics, as surveyed in chapters 9 and 10 of [1]. (See [14] and [20] for surveys on the VRP.)

Our generalization of the CIM neighborhood to the VRP has restrictions, and relies on heuristics. There are two fundamental difficulties that arise in any attempt to generalize the CIM to the VRP.

1. Proposition 1 of Section 2.1 fails to hold because of capacity and distance constraints. If we define independence of moves in a natural way, it will not, in general, be the case that a collection of independent moves leads to a feasible solution.
2. Proposition 2 of Section 2.1 relies on there being a complete order on cities. This property can be generalized, in multiple ways, and it is not obvious how to generalize it in the best way.

We address these difficulties as follows:

1. We will generalize the notion of "independence" of moves, and design a heuristic shortest path algorithm that simultaneously selects an independent set of moves and guarantees that capacity and distance constraints are satisfied.
2. We permit any partial order \prec of the cities satisfying the following relationship: if cities i and j are both in the same subtour, and if city i precedes city j , then $i \prec j$. This creates a partial order on the n cities. We then extend this partial order to a complete order, which is also denoted by \prec . In our improvement graph, if there is an arc (i, j) , then $i \prec j$.

Thompson and Psaraftis [28], developed an alternative approach for applying very large scale neighborhood search to the VRP based on the cyclic exchange neighborhood (see also, Thompson and Orlin [27], and Ahuja et al [4]). This technique also relies on a search of an appropriately defined improvement graph.

3.1 The Improvement Graph and the Search Algorithm

In this subsection, we introduce our notation for describing the routing T , its neighborhood, and the improvement graph. Let $T = (T_1, \dots, T_m)$ be a feasible routing. We let the set $N = \{1, 2, \dots, n\}$ denote the set of non-depot cities. We will refer to elements of N as cities, except when they appear in the improvement graph, in which case they will be called nodes.

For each city $i \in N$, $predCity(i)$ denotes the city that immediately precedes i on the subtour containing city i , and $sucCity(i)$ denotes the city that immediately follows i on its subtour. We say that the cities in N are *order-consistent* with T if $predCity(i) < i$ for each $i \in N$. This implies that $i < sucCity(i)$ with the exception of when $sucCity(i) = 0$.

We will construct an improvement graph with respect to a routing T . Before describing the improvement graph, we make a simplifying assumption.

Order Assumption: We assume that the cities in N are order-consistent with T .

We note that if the cities in N are not order consistent with T , then there are an exponential number of ways of relabeling the nodes of N so that they become order consistent. Of these ways, there are two relabelings that we give special attention to.

We say that the cities in N are *single tour ordered* if they are order consistent with T , and if $r < s$, then each city in T_r has an index less than each city in T_s . (We call this labeling "single

tour ordered” because we can view the sequence $1, 2, \dots, n$ as a tour, and if we ignore the depot, then each subtour occurs as consecutive cities on this tour.)

We say that the cities in N are *position ordered* if they are topologically ordered and if for $i < j$, the city in position i of subtour T_r has a label that is less than the city in position j of subtour T_s , for all r and s .

We will create three different VRP improvement graphs. The first graph will be referred to as the *ST-improvement graph*. For the ST-improvement graph, we assume that the nodes of N have been relabeled so that they are single tour ordered with respect to T .

We let $G = (V, E)$ denote the ST-improvement graph. The node set is

$$V = N \cup \{0_k : k = 1 \text{ to } m + 1\} \cup \{s\}.$$

There is a node in V for each node in N and for the first node (the depot) of each subtour. In addition, there is a node s for the origin node of the path and a final node 0_{m+1} . For each node $j \neq s$ and $j \neq 0_{m+1}$ we create two arcs (s, j) and $(j, 0_{m+1})$ with 0 cost. These arcs correspond to the null move.

We now define $pred(\)$ and $suc(\)$ for G as follows. If i is not the first non-depot node of its subtour, then $pred(i) = predCity(i)$. If i is the first non-depot node of subtour T_k , then $pred(i) = 0_k$. Similarly, if i is not the last non-depot node of its subtour, then $suc(i) = sucCity(i)$. If i is followed by the depot on T_k and $k < m$, then $suc(i) = 0_{k+1}$. There is an arc $(pred(i), i)$ for each node i in G , and the cost of this arc is 0. There are no arcs which imply moves involving the depot nodes.

For each pair $i, j \in N$, with $i < j$, $swap-move(i, j)$ represents the swapping of cities i and j . Thus i is deleted from its subtour, and j is inserted in its place. And j is deleted from its subtour and i is inserted in its place. For each $swap-move(i, j)$, we create an arc $(pred(i), suc(j))$ in the ST-improvement graph, and its cost $c^1(pred(i), suc(j))$ is the decrease in the objective function obtained by making the move. More precisely:

$$\begin{aligned} c^1(pred(i), suc(j)) = & -(d(pred(i), i) + d(i, suc(i)) + d(pred(j), j) + d(j, suc(j))) \\ & + (d(pred(i), j) + d(j, suc(i)) + d(pred(j), i) + d(i, suc(j))). \end{aligned}$$

For each $i, j \in N$, with $i < j$, $insertion-move(i, j)$ represents the deletion of node i from its subtour and its insertion in between cities j and $suc(j)$ in their subtour. For each $insertion-move(i, j)$,

we create an arc $(pred(i), suc(j))$ in the ST-improvement graph, and its cost $c^2(pred(i), suc(j))$ is the decrease in the objective function by making the move. More precisely:

$$\begin{aligned} c^2(pred(i), suc(j)) = & -(d(pred(i), i) + d(i, suc(i)) + d(j, suc(j))) \\ & + (d(pred(i), suc(i)) + d(j, i) + d(i, suc(j))). \end{aligned}$$

For each $i, j \in N$, with $i < j$, $reverse-insertion-move(i, j)$ represents the deletion of node j from its subtour and its insertion in between cities i and $pred(i)$ in their subtour. For each $reverse-insertion-move(i, j)$, we create an arc $(pred(i), suc(j))$ in the ST-improvement graph, and its cost $c^3(pred(i), suc(j))$ is the decrease in the objective function by making the move. More precisely:

$$\begin{aligned} c^3(pred(i), suc(j)) = & -(d(pred(i), i) + d(pred(j), j) + d(j, suc(j))) \\ & + (d(pred(i), j) + d(j, i) + d(pred(j), suc(j))). \end{aligned}$$

We let $c(i, j) = \min(c^1(i, j), c^2(i, j), c^3(i, j))$ for each $i, j \in V$, with $i < j$. The ST-improvement graph has $O(n)$ nodes, $O(n^2)$ arcs, and can be determined in $O(n^2)$ time.

We say that $swap-move(i, j)$ (respectively., $insertion-move(i, j)$ and $reverse-insertion-move(i, j)$) is *independent* of $swap-move(k, l)$ (respectively., $insertion-move(k, l)$ and $reverse-insertion-move(i, j)$) if $suc(j) < k$ or if $suc(l) < i$.

Theorem 7 *Suppose that T is a routing, and that the nodes are topologically ordered with respect to T . Let $G = (V, E)$ be an ST-improvement graph for the VRP. If P is a path in G from s to 0_{m+1} , then P corresponds to an independent set S of moves. Moreover, let $c(P)$ be the cost of path P and $d(T)$ be the total distance traveled in routing T . If the (possibly infeasible) routing T' is obtained from T by applying the moves in S , then $d(T') - d(T) = c(P)$.*

Proof We suppose P is a path from s to 0_{m+1} in G . Let P' be the set of edges obtained by deleting all edges of the form $(pred(i), i)$ in P . Then the edges of P' correspond to an ordered collection of moves such that for any two consecutive moves $move(i, j)$ and $move(k, l)$, $suc(j) < k$. Hence this collection of moves is independent, and its cost is equal to the cost of path P , $\sum_{(i,j) \in S} c(i, j)$, which is also equal to $d(T') - d(T)$. •

Theorem 7 states that any path in G from s to 0_{m+1} induces a set of compounded independent moves leading to a new routing T' . However, T' is possibly infeasible because it may violate the capacity constraints and it may violate the distance constraints for each subtour. Our way of obtaining feasible routings is to ensure that for each path P determined by the algorithm, we

obtain a feasible tour T' if we apply the compounded independent moves implied by P to T . We discuss how to maintain feasibility in subsection 3.3.

As for the TSP (see section 2.3), by weakening the independence requirement for compounding moves one can obtain larger sized neighborhoods for the VRP. We say that swap (respectively., insertion and reverse-insertion) $move(i, j)$ is *weakly independent* of swap (respectively., insertion and reverse-insertion) $move(k, l)$ if $suc(j) \leq k$ or $suc(l) \leq i$. The compounded weakly independent swap (respectively., insertion and reverse-insertion) neighborhoods can be searched by finding shortest paths with turn penalties on improvement graphs constructed similarly to the ones described above. The turn penalties are calculated in the same manner as in section 2.3.

3.2 Other Improvement Graphs

The *PO-improvement graph* is defined in exactly the same manner as the ST-improvement graph, except that at the beginning we relabel the nodes so that they are position ordered. Because the nodes have different labels, the improvement graph has a different set of arcs. All arcs (i, j) such that $i < j$ are present in the PO-improvement graph with a few exceptions. Arc (i, j) is not allowed if i and j have the same position in their respective subtours. Arcs of the form (i, j) are not allowed when $i = suc(k)$ and $j = pred(k)$ for some k or $suc(i)$ and j have the same position in their respective subtours. All arcs (i, j) such that i and $predCity(j)$ have the same position in their respective subtours imply null moves and have a cost of 0.

We also considered an enhancement to the swap neighborhood that is accomplished by exchanging consecutive nodes of cities. The improvement graph for the *multi-city swap* (MCS) neighborhood (we refer to this improvement graph as the MCS-improvement graph) is constructed as the ST-improvement graph with the addition of nodes representing a set of consecutive cities. For example, the node $\langle i, j \rangle$ represents the consecutive set of cities starting with node i in its subtour and ending with city j . (This presumes that i and j are in the same subtour, and that $i < j$.) A *swap-move* $(\langle i, j \rangle, \langle k, l \rangle)$ represents the deletion of the cities represented by $\langle i, j \rangle$ from its subtour and inserting the cities represented by $\langle k, l \rangle$ in between cities $pred(i)$ and $suc(j)$. At the same time, there is a deletion of the cities represented by $\langle k, l \rangle$ from its subtour and inserting the cities represented by $\langle i, j \rangle$ in between cities $pred(k)$ and $suc(l)$. In terms of the improvement graph, we let $pred(\langle i, j \rangle) = pred(i)$, and we let $suc(\langle i, j \rangle) = suc(j)$.

The arcs and costs for this improvement graph are defined in a similar manner to ST-improvement graphs. In our computational study we considered the pairs $\langle i, j \rangle$ for which $j - i \leq 4$.

We note that Theorem 7 and the implementation of compounded weakly independent moves directly extend to the PO-improvement graph and the MCS-improvement graph.

3.3 Partial Demands and Partial Capacities of Compounded Moves

Let $G = (V, E)$ be an improvement graph based on a routing T , and let P be any path in G with initial node 0_1 . Let $T^P = \{T_1^P, T_2^P, \dots, T_m^P\}$ be the routing obtained by applying the moves induced by P , as described in Theorem 7. We let $q(P, k) = \sum_{i \in T_k^P} q_i$. Thus, $q(P, k)$ is the amount of capacity used up on the k -th subtour after making the moves induced by P . Similarly, let $d(P, k) = \sum_{i \in T_k^P} (d(i, \text{succ}(i)) + b_i)$. Thus $d(P, k)$ is the distance accrued in the k -th subtour after making the moves induced by P . We let $d(P) = \sum_{k=1}^m \sum_{i \in T_k^P} d(i, \text{succ}(i))$. Thus $d(P)$ is the objective function value after making the moves induced by P .

We say that path P is *capacity feasible* if $q(P, k) \leq C_k$ for $k = 1$ to m . We say that path P is *distance feasible* if $d(P, k) \leq D_k$ for each k . In our heuristic, we will require that each path P that we maintain is both capacity feasible and distance feasible.

Suppose that $P = P' \cup (i, j)$, that is, P is obtained by appending arc (i, j) to path P' . Suppose that arc (i, j) represents a move that involves T_k and T_l . Then $d(P, r) = d(P', r)$ for $r \neq k$ or l , and $q(P, r) = q(P', r)$ for $r \neq k$ or l . Moreover, $d(P, k)$ can be computed from $d(P', k)$ in $O(1)$ time through an straightforward incremental calculation of the effect of the last move in P . Similarly, $d(P, l)$, $q(P, k)$, and $q(P, l)$ can be computed in $O(1)$ time incrementally from $d(P', l)$, $q(P', k)$, and $q(P', l)$.

3.3.1 The Heuristic Shortest Path Algorithm

The following algorithm is a variant of the dynamic programming algorithm for solving the shortest path problem on an acyclic graph. We heuristically ensure that the capacity and distance constraints are satisfied. We let $f(j)$ denote the distance label associated with node j for each $j \in V$. We let P_j denote the path from s to j determined by the algorithm.

We let $E(i)$ denote the arcs in E that emanate from the node i . We let $\text{parent}(i)$ be the node that precedes j on the path P_j .

Procedure Shortest Path

begin

$P_s := s;$

$f(s) := 0;$

```

for each  $r = 1$  to  $m$ , do  $d(P_s, r) := d(T_r)$  and  $q(P_s, r) := q(T_r)$ ;
for each  $j \in V$  such that  $j \neq s$ , do  $f(j) := c(s, j)$  and  $parent(j) := s$ ;
for each  $j \in V \setminus s$  do
  begin
    let  $i := parent(j)$ ;
     $P_j := P_i \cup (i, j)$ ;
    determine  $d(P_j, r)$  for all  $r = 1$  to  $m$ ;
    determine  $q(P_j, r)$  for all  $r = 1$  to  $m$ ;
    for each arc  $(j, k) \in E(j)$  do
      if  $f(k) > f(j) + c(j, k)$  and if the path  $P_j \cup (j, k)$  is both distance and capacity feasible
      then  $f(k) := f(j) + c(j, k)$  and  $parent(k) := j$ ;
    end
  end
end

```

Procedure Shortest Path is a multiple label keeping heuristic based on a label setting shortest path algorithm (see Ahuja et al. 1993 Chapter 4) for partially searching ST, PO, and MCS improvement graphs. Such multiple label keeping heuristics are widely used in the literature for solving constrained shortest paths problems (see for example [11]).

Theorem 8 *Procedure Shortest Path determines a path P_j from s to j for all j such that P_j is both distance and capacity feasible. The time to run the shortest path procedure is $O(n^2)$.*

Proof There is a trivial path consisting of the arc (s, j) . Moreover, any other path P_j from s to j obtained by the algorithm is guaranteed to be distance and capacity feasible. We now consider the computation time.

Immediately before the arcs in $E(j)$ are scanned in the for loop, we need to determine $d(P_j, r)$ for all r , and we need to determine $q(P_j, r)$ for all r . In the algorithm, $P_j = P_i \cup (i, j)$. The values $d(P_i, r)$ and $q(P_i, r)$ have already been calculated for all r prior to this step. So, $d(P_j, r)$ and $q(P_j, r)$ can be calculated incrementally in $O(1)$ step, and all values can be calculated in $O(nm)$ steps. Since the number m of vehicles is less than the number n of cities, this is dominated by $O(n^2)$.

Each arc (j, k) is scanned once in the inner for loop. When an arc is scanned, we need to determine whether the path $P_k = P_j \cup (j, k)$ is distance and capacity feasible. By the above, we can determine feasibility incrementally from $d(P_j, \cdot)$ and $q(P_j, \cdot)$ in $O(1)$ steps. Thus the time to scan

each arc and determine its feasibility is $O(1)$ steps. Since there are $O(n^2)$ arcs, the running time is $O(n^2)$ steps in total, completing the proof. •

The Procedure Shortest Path always finds a feasible path on the ST, PO, and MCS improvement graphs with a non-positive cost. However, to embed our algorithms within a tabu search framework, our algorithms need to identify moves with minimum positive cost if the Procedure Shortest Path fails to find a feasible improving path. Hence we modify the improvement graphs for the CWIM neighborhoods. For each city $j \neq s$ and $j \neq 0_{m+1}$ we create arcs (s, j') and eliminate arcs (s, j) , and we create arcs $(j, 0_{m+1})$ and eliminate arcs $(j', 0_{m+1})$. These arcs correspond to null moves and have 0 cost, however they do force the Procedure Shortest Path to find a path that corresponds to at least one non-null move.

4 Computational Study for the VRP

In this section we provide the results of a computational study of the CIM neighborhood search algorithm embedded into a simple tabu search framework. The purposes of this computational study on the VRP are:

- To provide an understanding of the computational power of the proposed algorithms by comparing the performance of our algorithms to that of the state of the art tabu search algorithms.
- To understand the sources of the algorithms' merits and bottlenecks.

In our computational experiments we primarily use two sets of standard test problems for the VRP and the VRP with distance constraints. The first set is the 14 instances described in [6] and obtained from the ORLIB at <http://mscmga.ms.ic.ac.uk/jeb/orlib/vrpinfo.html>. We use these instances for (i) setting the parameters in our tabu search framework (ii) comparing the performance of various compounded and simple move neighborhoods (iii) comparing the performance of the CWIM neighborhood search algorithm to the performance of the best tabu search heuristics from the literature. The second set of test problems are the larger instances described in [18]. We compare the results obtained on these instance by our large scale neighborhood search algorithms to the results reported in [18] and [29].

The experimental design framework and the values of the parameters we used for obtaining the results throughout this paper are as follows:

1. We used a randomized version of the Clark and Wright (CW) algorithm [7] to construct our initial solutions. The Clark and Wright algorithm is a well-known construction heuristic that produces solutions to the VRP in $O(n^2)$ time. In general, the CW algorithm creates good solutions in which each vehicle’s capacity is tightly utilized. During our initial experimentation we found that this tightness left very little room for our local search algorithm to make moves. Therefore, to obtain initial random solutions that are less tight in terms of their capacity and distance restrictions we used the CW algorithm on a modified instance of the VRP in which the capacity of each vehicle is reduced by a random percentage less than r_C and the distance constraint is reduced by a random percentage less than r_D . After our initial experiments we set $r_C = 20\%$ and $r_D = 30\%$ and kept it constant throughout the computational study.
2. After the initial solution is created, we search for a compounded move by constructing an improvement graph and heuristically finding a capacity and distance feasible shortest path on this graph. We searched for the best compounded weakly independent moves based on the ST, PO, and MCS improvement graphs of Section 3. At each iteration we searched the CWIM neighborhood based on one of the three improvement graphs selected randomly. Now we give some particulars of the search based on each improvement graph:
 - (a) For the ST-improvement graph, the ordering of the vehicles affects the graph and this affects the neighborhood searched. In our heuristic we kept a fixed ordering of the vehicles; however we chose the first vehicle at random every time an ST-improvement graph is generated. Note that this set up allows for any city from any location on one vehicle to move to any location on a following vehicle’s tour.
 - (b) For the PO-improvement graph, we considered an improvement graph based on the current routing $T = (T_1, T_2, \dots, T_m)$. Let $T^{-1} = (T_1^{-1}, T_2^{-1}, \dots, T_m^{-1})$ be a routing where the cities in each tour T_k^{-1} is visited in exactly the opposite order of the tour T_k . Assuming the distances between cities are symmetric, if routing T is feasible then routing T^{-1} is also feasible. We considered an alternative PO-improvement graph based on the routing T^{-1} . In our heuristic, whenever we generate a PO-improvement graph, we randomly choose one of the two graphs.
 - (c) For the MCS-improvement graph we allowed for swapping of up to 5 consecutive cities.
3. If the shortest path in the current improvement graph has a negative cost, then the move

implied by this shortest path is applied to the current solution and the improvement graph is updated for the next iteration. However, if the shortest feasible path found has non-negative cost, then our heuristic was unable to find an improved solution. We allow some non-improving moves by incorporating the exchanges implied by the positive cost shortest path found. If the *best* solution found so far for a given initial solution is not improved for β iterations we restart the search with a new solution generated by our randomized version of the CW algorithm. In our experiments we set $\beta = 1,000$ for the vehicle routing problems, and set $\beta = 8,000$ for the VRP with distance constraints.

4. Since moves with positive costs are allowed, in order to prevent cycling, we embed the neighborhood search heuristic into a very simple tabu search mechanism. If a node is moved from its current location in the latest compounded move, it is made tabu for z iterations of the algorithm. When a node is made tabu it is not allowed to be moved in an exchange by the elimination of its corresponding improvement graph arcs. The values for z are chosen at random from the interval $[z_{\min}, z_{\max}]$. For generating the experimental results in this proposal, we let $[z_{\min}, z_{\max}] = [4, 9]$.

In our implementation we run our algorithms for the 14 test problems algorithm on a Pentium III computer with 256 KB cache, 256 MB RAM, and 733 MHz processor speed and for the larger instances on a SunBlade workstation with 256 KB cache, 256 MB RAM, and 733 MHz processor speed.

In the rest of this section we compare our algorithm with other tabu search heuristics, where the stopping criteria for our heuristic is a *time limit*. Then we demonstrate results from our algorithm when the stopping criterion is a *combinatorial count*. This way we gauge the results from the time limited runs and present reproducible results. Furthermore, for each problem, we give the best-ever results that we produced with various sets of parameters over the course of our experimental study. Moreover, we present our results on the larger 20 test problem, and conclude with a study in which we compare the results obtained by various single and compounded moves neighborhoods.

4.1 Comparison with the state of the art tabu search heuristics

In Table 1 we compare the solutions generated by our algorithm with the solutions generated by several competitive tabu search heuristics. We use the 14 problems from [6] for this comparison .

The problems are VRP instances all with capacity and some with distance restrictions. We first discuss the other tabu search algorithms briefly.

“Taburoute” developed by Gendreau et al in [15] uses single insertion moves. Given a node, Taburoute uses a sophisticated methodology to choose a good tour for it to be inserted into. Two other features of the algorithm are that it allows for infeasible solutions to be searched and that periodically the tour in which a vertex has just been inserted in is re-optimized. The authors report two different results for the algorithm: (i) Results with a standard set of parameters for every problem. We present them in column TRSP. (ii) The best results obtained for each problem using different parameters. We present these results in column TRBP. The times for TRSP are given for computations on a Silicon Graphics workstation, 36 MHz, 5.7 Mflops.

Tailard’s tabu search algorithm in [26] produces some of the best results for the 14 problems we consider in this section. The algorithm uses both insertion and swap moves. Also, only the creation of feasible tours are allowed, and occasionally individual tours are re-optimized. A novel feature of the algorithm is that it is suitable for decomposition into subproblems. Hence Tailard uses parallel computing to generate solutions with values given in column TA of Table 1. However their overall computation times are not reported.

We also report the results from a parallel implementation by Rego and Roucairol [25] of an ejection chain based tabu search algorithm in column RR. The times for the algorithm are given for computations on a Sun Sparc4 IPC, 2.5 - 4.3 Mflops.

We report the results obtained by the Xu and Kelly tabu search algorithm which is based on applying a few simple swaps and insertion moves to the current solution. The results we report in column XK are based on computations performed on a DEC Alpha computer with 26-43 Mflops and are obtained from [18].

Finally we report the results by Toth and Vigo [29] on their Granular Tabu Search (GTS) algorithm. The GTS approach is based on using smaller neighborhoods that include moves that are more likely to lead to improved solutions. Philosophically, the GTS approach to solve hard problems is in the opposite direction of the VLSN search approach. The results reported in column GTS are based on computations on a Pentium 200 MHz PC, 15 Mflops.

In our implementation we use a standard set of parameters and run the algorithm for 5000 cpu seconds on a Pentium III computer with 256 KB cache, 256 MB RAM, and 733 MHz processor speed. At each iteration we randomly choose one of the three improvement graphs described in Section 3 to create a neighborhood and to search it. The parameters are set to the values described

in the above section. We give the average value obtained over five runs together with the average time the algorithm used to reach these values, in column CIMA. We also give the best value we obtained in these five runs and the time used for reaching this value in column CIMB. Whenever we obtain the best value more than once, we report the best time to reach this value.

In Table 1, the first column presents the number of nodes n , number of minimum vehicles m needed, reference r to the first paper that describes the instance, and the problem type, Y . Reference e indicates instances proposed in [5] and c indicates the instances from [6]. Type C indicates that the problem has capacity constraints, and type D indicates that the problem has both capacity and distance constraints. The second column gives the value of the best known solution for the specific problem. The rest of the columns represent the results of the various algorithms as described above. The first row gives the percentage deviation from the best reported solution and the second row gives the computational time in cpu minutes whenever available. Also, note that real distances are used.

From these results we can conclude that the Tailard’s algorithm (TA) dominates the other ones. For the TA algorithm the computational times for reaching the final solution can not be compared as they are not reported. As an indication of computational times for the TA algorithm we report the times required to reach a prescribed solution quality for the seven capacitated VRP instances in Table 2. Among the rest of the algorithms the Granular Tabu Search, Taburoute, and the Compounded Weakly Independent Moves algorithms are the competitive ones. The TRBP seems to dominate the CIMA. The results for Taburoute with a standard set of parameters are dominated by the CIMA results. Also in Table 4 where we report the best values obtained by the CIM algorithm over the course of our computational study with various set of parameters, we see that the dominance of the TRBP disappears. In comparing GTS and CIMA, we note that GTS is very fast and creates high quality solutions for most of the instances, however for the larger problems although slow CIM neighborhood search creates solutions that are up to 2% better. Hence we can conclude that although not very fast, the CIM algorithm is a competitive algorithm for the VRP with capacity and distance restrictions.

4.2 Computations on the large instances

In this section, we report the performance of the CIM heuristic on the 20 new large-scale instances introduced in [18]. We compare the results of the CIM heuristic with the results of GTS, XK, and with a deterministic heuristic, RTR, described in [18]. The results of the XK algorithm on

n, m, r, Y	Best	CIMA	CIMB	TRSP	TRBP	TA	XK	RR	GTS
50, 05, e, C	524.61	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
		36.92	23.13	6.00	-	-	29.92	0.85	0.81
75,10, e, C	835.26	100.12	100.02	100.06	100.01	100.00	100.00	100.27	100.40
		37.84	33.93	53.80	-	-	48.80	16.80	2.21
100,08, e, C	826.14	100.25	100.16	100.40	100.00	100.00	100.00	100.17	100.29
		37.10	21.30	18.40	-	-	71.93	33.90	2.39
100,10, c, C	819.56	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
		16.00	5.98	16.00	-	-	56.61	1.22	1.10
120, 07, c, C	1042.11	100.06	100.00	103.01	100.00	100.00	100.00	100.14	100.07
		43.80	69.13	22.20	-	-	91.23	6.30	3.18
150, 12, c, C	1028.42	100.79	100.76	100.75	100.26	100.00	100.11	102.52	100.47
		35.43	24.45	58.80	-	-	149.90	16.25	4.51
199, 17, c, C	1291.45	101.43	101.23	102.42	101.54	100.57	100.55	103.64	102.08
		38.94	57.25	90.90	-	-	272.52	16.25	7.50
Ave. dev. Type C		100.38	100.31	100.95	100.26	100.08	100.09	100.96	100.47
Ave. time		35.15	33.57	38.01	-	-	102.99	14.65	3.10
50, 06, c, D	555.43	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
		30.43	3.5	13.50	-	-	30.67	3.17	0.86
75, 11, c, D	909.68	100.11	100.04	100.39	100.01	100.00	106.15	100.00	101.21
		31.83	36.53	54.60	-	-	102.13	23.10	2.75
100, 09, c, D	865.94	100.00	100.00	100.40	100.00	100.00	101.78	100.27	100.41
		20.54	12.43	25.60	-	-	98.15	8.60	2.90
100, 11, c, D	866.37	100.00	100.00	100.00	100.00	100.00	105.64	100.02	100.00
		41.83	6.55	65.70	-	-	152.98	9.42	1.41
120, 11, c, D	1541.14	100.35	100.25	102.12	100.31	100.00	105.02	100.59	100.28
		57.03	39.73	59.20	-	-	201.75	2.00	9.34
150, 14, c, D	1162.55	100.40	100.20	101.31	100.03	100.00	-	101.40	100.91
		47.58	42.47	71.00	-	-	-	15.55	5.67
199, 18, c, D	1291.45	100.69	100.61	101.62	100.35	100.15	103.11	101.79	102.86
		38.65	28.32	99.80	-	-	368.37	52.02	9.11
Ave. dev. Type D		100.22	100.10	100.78	100.18	100.02	103.62	100.58	100.81
Ave. time		38.65	28.32	55.63	-	-	160.30	16.26	4.58
Ave. dev. Overall		100.30	100.24	100.86	100.18	100.05	101.72	100.77	100.64
Ave. time		36.90	30.95	46.82	-	-	131.65	15.45	3.84

Table 1: Computational comparisons with other tabu search heuristics.

n, m, r, Y	TA	TA
	5%	1%
50, 05, e, C	0.12	0.82
75, 10, e, C	0.05	0.88
100, 08, e, C	0.20	9.67
100, 10, c, C	1.35	5.67
120, 07, c, C	76.67	-
150, 12, c, C	1.43	63.33
200, 17, c, C	1.25	50.00
Average	11.58	21.73

Table 2: Time in minutes for the TA algorithm to reach a prescribed solution quality.

the capacity and distance restricted VRP are not presented because the results reported in [18] are obtained with an incorrect distance matrix. To obtain the results on these instances, the CIM algorithm was run on a SunBlade100 workstation with an 500 MHz CPU UltraSparcIIe processor, the RTR algorithm was run on a Pentium 100 MHz PC, GTS and XK were run on the same machines as described above.

In Table 3, the results from CIM algorithm are reported for a variable number of vehicles. That is, we present the best results we obtain without assuming that there is a constraint on the number of available vehicles. For accuracy, in the CIM Solution column we report the best value we obtain together with the number of vehicles that are needed in our solution. Note that here we are solving a different problem from the other algorithms; hence, the results should be compared with this in mind. For example, if the CIM algorithm is able to improve the best solution distance by 2% while using one more vehicle than that is necessary, then the result could be interpreted as 'A 2% improvement can be obtained by increasing the number of vehicles by one'. The results for the CIM algorithm are the best of two runs, where each run is terminated after 417 minutes. In the CIM Time column we present the minutes required to reach the reported solution.

The results in Table 3 present a mixed judgment on the performance of the CIM algorithms. On the capacity restricted problems, the CIM algorithms usually use more than the necessary number of vehicles, and find solutions that are between 98.85% and 100.44% of the best solution value for the constant vehicle number problem.

For the 200 city problem, the CIM algorithms found a solution that improves the best solution generated by any of the algorithms GTS, XK, or, RTR by approximately 2% while using one more vehicle, then the minimum obtainable. For the 240 city problem, CIM algorithms generated a

n, m, Y	Best	CIM Sol.	CIM	GTS	XK	RTR
240, 22, C	720.44	716.46 - 25	99.44 223.62	98.70 14.29	103.72 2314.00	100.00 5.69
252, 27, C	881.04	870.90 - 33	98.85 235.13	98.61 11.43	100.00 1465.77	100.00 6.01
255, 14, C	587.09	587.89 - 15	100.14 49.20	101.07 11.67	100.34 340.20	100.00 23.01
300, 28, C	1029.21	1023.32 - 34	99.43 299.23	98.80 21.45	103.63 4101.02	100.00 8.15
320, 30, C	1103.69	1097.11 - 32	99.40 31.17	99.32 14.51	101.30 1577.30	100.00 21.83
323, 16, C	746.56	749.85 - 20	100.44 125.05	100.68 15.83	100.00 501.82	100.35 31.49
360, 33, C	1403.05	1404.84 - 38	100.13 393.03	99.85 30.06	102.34 5718.38	100.00 12.42
396, 34, C	1364.23	1367.15 - 37	100.21 65.30	100.38 18.45	100.99 4340.07	100.00 32.62
400, 18, C	932.68	932.74 - 21	100.01 171.05	100.36 33.12	100.00 852.72	100.18 69.19
420, 41, C	1875.17	1883.33 - 43	100.44 121.62	102.17 43.05	103.19 10839.73	100.00 31.05
480, 38, C	1656.66	1643.00 - 39	99.18 31.58	99.74 23.07	100.00 8943.45	100.08 47.55
483, 19, C	1137.18	1134.63 - 22	99.78 388.62	100.88 42.90	100.31 1151.10	100.00 101.09
200, 05, D	6702.73	6570.28 - 06	98.02 15.50	99.92 2.38	- 591.40	100.00 11.24
240, 10, D	5834.60	5741.79 - 11	98.41 134.95	98.31 4.98	- 802.87	100.00 3.68
280, 08, D	9016.93	8836.25 - 08	98.00 81.98	99.41 4.65	- 913.70	100.00 18.79
320, 10, D	8566.04	8917.41 - 13	104.10 150.83	99.85 8.28	- 898.53	105.09 22.66
360, 09, D	11047.69	11116.68 - 10	100.62 85	95.47 11.66	- 1062.73	101.50 22.55
400, 10, D	11649.06	12106.64 - 14	103.93 15.67	97.89 12.94	- 1749.27	101.98 40.04
440, 11, D	12250.06	12634.17 - 13	103.14 33.95	98.25 11.08	- 1586.20	102.16 111.37
480, 12, D	14639.92	15316.69 - 12	104.63 106.50	101.85 15.13	- 2432.42	100.00 122.61

Table 3: Computational comparisons on the twenty large-scale instances. The number in the first row in each sell is the percentage deviation from the best solution and the number in the second row is the computing time in minutes.

solution, again using one additional vehicle, which is competitive with the best solution obtained by the GTS algorithm. Finally, for the 280 city problem, CIM algorithms improved the best solution of the GTS algorithm, using the same number of vehicles, by 1.42%.

The quality of the results that CIM algorithms obtain for the remaining five distance restricted VRP is lower. The solution values are worse than the best solution by a value between 0.66% and 4.63%. This decrease in quality surprised us given the good performance of the CIM algorithms on the seven distance restricted instances of the previous section and on the rest of the twenty large-scale instances presented in this section. In terms of the computational times, the GTS algorithm is the most efficient methodology of those described here. The running time of the CIM algorithm is a large constant times the running time of the GTS algorithm.

4.3 Analyzing the performance of the compounded independent moves algorithm

In this section, we try to further understand the performance of the CIM algorithm by examining computational data on the same 14 test problems of Section 4.1. We first compare the results from two experiments: (i) data from computations where the algorithm stops when it reaches a threshold as number of neighbors created, and (ii) data from computations where the stopping criterion is a bound on the running time.

Using a combinatorial count, such as the number of neighbors generated, as the stopping criterion permits the computational results to be reproducible across computation platforms, and across different implementations of data structures. We also use this data to verify the computational growth of the algorithms, as discussed in Section 3. For the rest of the paper, we will use the number of neighbors generated as the stopping criterion.

We give a set of best values produced over the course of our experimental studies with various parameter settings. Our goal is to present results that provide insight into the sensitivity of the algorithms to these parameter settings.

In Table 4 column CC, we present results produced when the algorithm is set to run until 500,000 neighbors are generated. We give the results in terms of percentage deviation from the best known solution and cpu seconds used to generate and search 500,000 neighbors. In column TLA, we give the average deviation from the best known values and the average number of neighbors generated over five runs of the algorithm when it is let to run for 5000 seconds. The last column, Best, presents the best results we obtained with various parameter settings when the run again was

terminated after 5000 seconds.

Comparing the Best values with the values obtained from the other two runs with a standard set of parameters, we conclude that the algorithm is not very sensitive to the parameter settings. The largest improvement in the percentage deviation from optimality is 0.28 going from the standard set of parameters to specifically adjusted ones. This is not a wide gap when put in perspective by comparing it to improvements up to 3.00% between the percentage deviations of the Taburoute algorithm with standard parameters versus best parameters (see for example the values for problem 120, 07, c, C on Table 1).

Next, we analyze the effects of different features of the compounded moves algorithm. We use the same set of 14 problems as above and present the results from two comparative studies. First, we compare the performance of the algorithm (presented in column All_s5) with the settings described in the beginning of Section 4 with the algorithms:

- Allow only the ST and PO improvement graphs based neighborhoods (All_s1),
- Allow only the PO-improvement graph based neighborhoods (PO),
- Allow only the MCS-improvement graph based swapping of up to groups of five cities (MCS_s5),
- Allow only the ST-improvement graph based neighborhood (ST).

Then we compare the performances of the algorithms All_s5, MCS_s5, and PO with the same neighborhoods when only a single move is allowed (that is instead of a combination of moves) at each iteration.

In Table 5 we present the performance of the algorithm with the five different compounded moves neighborhoods mentioned above. For each neighborhood, we run the algorithm three times, generating a different initial solution each time, until 500,000 neighbors are generated. The results in the table represents the average over these three runs. The running time is the time in seconds it took for the algorithm with the particular neighborhood to generate and search 500,000 neighbors. Table 5 implies that using all three improvement graphs PO, MCS, and ST together has an important effect on the performance of the algorithm. Although the MCS and the ST perform (see columns MCS_s5 and ST) much worse on their own, combining them with the PO based neighborhood significantly improves the performance of the PO in terms of percentage deviation from the best known solution (see columns All_s5 and PO). Furthermore, the running time, for generating 500,000 neighborhoods, of the MCS_s5 is much worse then the running time of the other

n, m, r, Y		CC	TLA	Best
50, 05, e, C	% deviation	0.00	0.00	0.00
	running time	448	5,000	5,000
	# neighbors	500,000	5,768,098	
75, 10, e, C	% deviation	0.19	0.12	0.01
	running time	883	5,000	5,000
	# neighbors	500,000	2,824,446	
100, 08, e, C	% deviation	0.28	0.25	0.00
	running time	2,055	5,000	5,000
	# neighbors	500,000	1,277,031	
100, 10, c, C	% deviation	0.00	0.00	0.00
	running time	2,245	5,000	5,000
	# neighbors	500,000	1,133,189	
120,07, c, C	% deviation	0.10	0.06	0.00
	running time	2,482	5,000	5,000
	# neighbors	500,000	896,113	
150, 12, c, C	% deviation	0.51	0.79	0.30
	running time	4,262	5,000	5,000
	# neighbors	500,000	602,782	
199, 17, c, C	% deviation	1.04	1.43	0.75
	running time	7,562	5,000	5,000
	# neighbors	500,000	363,960	
50, 06, c, D	% deviation	0.00	0.00	0.00
	running time	458	5,000	5,000
	# neighbors	500,000	4,945,203	
75, 11, c, D	% deviation	0.24	0.11	0.04
	running time	983	5,000	5,000
	# neighbors	500,000	2,567,459	
100, 09, c, D	% deviation	0.00	0.00	0.00
	running time	2,062	5,000	5,000
	# neighbors	500,000	1,194,039	
100, 11, c, D	% deviation	0.00	0.00	0.00
	running time	2,753	5,000	5,000
	# neighbors	500,000	766,830	
120, 11, c, D	% deviation	0.20	0.34	0.21
	running time	3,727	5,000	5,000
	# neighbors	500,000	766,830	
150, 14, c, D	% deviation	0.08	0.40	0.06
	running time	5,776	5,000	5,000
	# neighbors	500,000	456,893	
199, 18, c, D	% deviation	0.73	0.69	0.61
	running time	10,055	5,000	5,000
	# neighbors	500,000	279,191	

Table 4: Running time vs. number of neighborhoods generated.

algorithms. This is due to the increased size of the improvement graph with the addition of the nodes representing the groups of cities (from 1 to 5 city groups). Given this information, we ran the algorithm with all the neighborhoods based on PO, and ST improvement graphs. Results of these runs (see columns All_s5, and All_s1) show that under these settings although the running time improves, the performance of the algorithm gets worse.

In Table 6, we compare the performance of the algorithm when the PO, MCS, and ST neighborhoods are all used with compounded moves (column All_s5) versus the performance of the algorithm when the same combination of the neighborhoods are used with only single moves (column Single_All_s5). The results in Table 6 are again averages from three runs, and each run is terminated after 500,000 neighbors are created. For searching the combined single moves neighborhood, we use the same improvement graph structure except that we delete the edges that allow to start a new move when a move is completed. The running times of the algorithms are in the same order of magnitude. On average the percentage deviation from optimality for both of the algorithms does not differ significantly. Moreover, combining this finding with the results from Table 5, we conclude that combining different neighborhoods has a more significant effect on the performance than compounding moves.

We further analyze this conclusion by comparing the performances of algorithms PO and MCS with compounded moves versus with single moves. In Table 7, we compare the performance of the PO algorithm using compounded moves (column PO) with its performance using single moves (column PO_Single). Again the conclusions are similar to above, that is, the performances in terms of percentage deviation from optimality on average are very similar (in this case over the specific three runs the average is actually identical). Nevertheless, the compounded moves PO performs better than the single moves PO neighborhood for the largest problems with 199 cities. Finally, we compare the performance of compounded moves (column MCS) versus single moves MCS (column MCS_Single) neighborhoods in Table 8. Although the compounded moves MCS perform slightly better on most of the problems, we similarly conclude that on average the results for these two algorithms are not significantly different.

5 Summary

We study compounded independent moves neighborhood search algorithms for the vehicle routing problem with capacity and distance restrictions. We show that large-scale compounded moves neighborhoods can be created and searched efficiently via network flow techniques.

n, m, r, Y		All_s5	All_s1	PO	MCS_s5	ST
50, 05, e, C	% deviation	0.00	0.00	0.00	0.04	0.02
	running time	551	264	254	1,816	216
75, 10, e, C	% deviation	0.18	0.70	0.28	2.42	2.14
	running time	1,121	1,108	559	4,008	506
100, 08, e, C	% deviation	0.29	0.48	0.38	2.94	0.71
	running time	1,945	2,177	1,105	8,006	1,148
100, 10, c, C	% deviation	0.00	0.00	0.00	4.50	0.19
	running time	2,818	1,632	1,261	8,985	1,119
120, 07, c, C	% deviation	0.16	0.47	0.29	2.23	1.04
	running time	3,070	2,921	1,404	14,045	1,328
150, 12, c, C	% deviation	0.82	1.47	1.99	5.45	1.98
	running time	4,443	3,501	2,451	9,032	2,532
199, 17, c, C	% deviation	1.22	2.27	2.16	5.68	3.09
	running time	7,153	4,649	4,354	36,320	4,483
50, 06, c, D	% deviation	0.02	0.23	0.13	4.19	0.84
	running time	584	368	391	2,174	358
75, 11, c, D	% deviation	0.27	0.66	0.32	5.31	1.22
	running time	1,258	687	1,564	8,945	1,622
100, 09, c, D	% deviation	0.00	0.00	0.00	6.42	1.40
	running time	2,179	2,422	1,894	10,333	1,862
100, 11, c, D	% deviation	0.00	0.00	0.01	6.08	0.29
	running time	3,099	2,715	2,931	2,575	2,981
120, 11, c, D	% deviation	0.34	1.00	0.82	1.85	2.80
	running time	3,556	1,973	2,608	15,751	1,840
150, 14, c, D	% deviation	0.33	0.81	1.12	7.01	2.81
	running time	6,499	8,004	7,684	41,336	7,945
199, 18, c, D	% deviation	0.76	1.17	1.93	10.11	3.43
	running time	16,622	6,713	9,664	42,842	9,373
	Ave. % deviation	0.31	0.66	0.67	4.59	1.57

Table 5: Effect of the different compounded moves neighborhoods.

n, k, r, T	All_s5	Single_All_s5
50, 05, e, C	0.00	0.00
75, 10, e, C	0.18	0.28
100, 08, e, C	0.29	0.24
100, 10, c, C	0.00	0.00
120, 07, c, C	0.16	0.16
150, 12, c, C	0.82	0.72
199, 17, c, C	1.22	1.29
50, 06, c, D	0.02	0.15
75, 11, c, D	0.27	0.17
100, 09, c, D	0.00	0.00
100, 11, c, D	0.00	0.00
120, 11, c, D	0.34	0.27
150, 14, c, D	0.33	0.30
199, 18, c, D	0.76	0.83
Ave. % deviation	0.31	0.32

Table 6: Compounded moves versus single moves.

n, k, r, T	PO	Single_PO
50, 05, e, C	0.00	0
75, 10, e, C	0.28	0.18
100, 08, e, C	0.38	0.31
100, 10, c, C	0.00	0.00
120, 07, c, C	0.29	0.03
150, 12, c, C	1.99	1.97
199, 17, c, C	2.16	2.34
50, 06, c, D	0.13	0.13
75, 11, c, D	0.32	0.44
100, 09, c, D	0.00	0.00
100, 11, c, D	0.01	0.00
120, 11, c, D	0.82	0.79
150, 14, c, D	1.12	1.05
199, 18, c, D	1.93	2.13
Ave. % deviation	0.67	0.67

Table 7: Compounded insertions and reverse-insertions versus single insertions and reverse-insertions searched via the PO improvement graph.

n, k, r, T	MCS	Single_MCS
50, 05, e, C	0.04	0.05
75, 10, e, C	2.42	2.65
100, 08, e, C	2.94	3.34
100, 10, c, C	4.50	1.32
120, 07, c, C	2.23	2.04
150, 12, c, C	5.45	5.06
199, 17, c, C	5.68	5.98
50, 06, c, D	4.19	4.21
75, 11, c, D	5.31	4.74
100, 09, c, D	6.42	7.72
100, 11, c, D	6.08	7.02
120, 11, c, D	1.85	1.52
150, 14, c, D	7.01	7.07
199, 18, c, D	10.11	10.55
Ave. % deviation	4.59	4.52

Table 8: Compounded swaps versus single swaps searched via the MCS improvement graph.

Our computational study on the classical fourteen test problems and the twenty large-scale instances is able to show that the compounded moves neighborhoods are competitive approaches for solving vehicle routing problems. Furthermore, we present the results of a set of computational tests geared towards analyzing the strengths and bottlenecks of our approach.

Acknowledgments

This research was supported by NSF Grant DMI-9820998.

References

- [1] E. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimization*, (John Wiley & Sons, New York, 1997).
- [2] R.K. Ahuja, Ö. Ergun, J.B. Orlin, and A.B. Punnen, A survey of very large-scale neighborhood search techniques, *Discrete Applied Mathematics* 123 (2002) 75-102.
- [3] R.K. Ahuja, J.B. Orlin, and Thomas L. Magnanti, *Network Flows*, (Prentice-Hall, NJ, 1993).
- [4] R.K. Ahuja, J.B. Orlin, and D. Sharma, Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem, *Mathematical Programming* 91 (2001) 71-97.
- [5] N. Christofides and S. Eilon, An algorithm for the vehicle dispatching problem, *Operations Research Quarterly* 20 (1969) 309-318.
- [6] N. Christofides, A. Mingozzi, and P. Toth, The vehicle routing problem. N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, eds., *Combinatorial Optimization*, (Wiley, Chichester, 1979).
- [7] G. Clark and J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research* 12 (1964) 568-581.
- [8] R.K. Congram, Polynomically searchable exponential neighborhoods for sequencing problems in combinatorial optimization, *Faculty of Mathematical Studies Thesis*, University of Southampton, 2000.
- [9] R.K. Congram, C.N. Potts, S. L. van de Velde, An iterated dynasearch algorithm for the single machine total weighted tardiness scheduling problem, *INFORMS Journal on Computing* 14 (2002) 52-67.
- [10] V. Deineko and G.J. Woeginger, A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem, *Mathematical Programming Series A* 87 (2000) 519-542.
- [11] M. Desrochers and F. Soumis, A generalized permanent labeling algorithm for the shortest path problem with time windows, *INFOR* 26 (1988) 191-212.
- [12] Ö. Ergun, New neighborhood search algorithms based on exponentially large neighborhoods, *Operations Research Center Thesis*, MIT, 2001.

- [13] R.T. Firla, B. Spille and R. Weismantel, personal communication.
- [14] M. Fisher, Vehicle routing. M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, eds., Network Routing, Handbooks in OR & MS Vol. 8, (Elsevier, Amsterdam, 1995).
- [15] M. Gendreau, A. Hertz, and G. Laporte, A tabu search heuristic for the vehicle routing problem, *Management Science* 40 (1994) 1276-1290.
- [16] M. Gendreau, G. Laporte, and J.Y. Potvin, Vehicle routing: modern heuristics, E. Aarts and J.K. Lenstra eds., *Local Search in Combinatorial Optimization*, (John Wiley & Sons, New York, 1997) 311-336.
- [17] F. Glover, Ejection chains, reference structures, and alternating path algorithms for the traveling salesman problem, Research Report, University of Colorado-Boulder, Graduate School of Business, 1992. {A short version appeared in *Discrete Applied Mathematics* 65 (1996) 223-253.}
- [18] B.L. Golden, E.A. Wasil, J.P. Kelly, and I. Chao, The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results, T.G. Crainic and G. Laporte eds., *Fleet Management and Logistics*, (Kluwer Academic Publisher, Boston, 1998) 215-310.
- [19] R.F. Kirby and R.B. Potts, The minimum route problem for networks with turn penalties and prohibitions, *Transportation Research* 3 (1969) 397-408.
- [20] G. Laporte and I.H. Osman, Routing problems: a bibliography, in *Freight Transportation*, G. Laporte and M. Gendreau, eds., *Annals of Operations Research* 61 (1995) 227-262
- [21] C.L. Liu, *Elements of Discrete Mathematics*, (McGraw-Hill, Singapore, 1985).
- [22] C.N. Potts and S.L. van de Velde, Dynasearch - Iterative local improvement by dynamic programming: Part I, The traveling salesman problem, Technical Report, University of Twente, The Netherlands, 1995.
- [23] A.P. Punnen and F. Glover, Ejection chains with combinatorial leverage for the TSP, Research Report, University of Colorado-Boulder, 1996.
- [24] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter, Jr., On the capacitated vehicle routing problem, Research Report (2000). Updates available at <http://www.branchandcut.org>.

- [25] C. Rego and C. Roucairol, A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In I.H. Osman and J.P. Kelly eds., *Meta-Heuristics: Theory & Applications*, (Kluwer Academic Publisher, Boston, MA, 1996) 661-675
- [26] E. Taillard, Parallel iterative search methods for vehicle routing problem, *Networks* 23 1993661-673.
- [27] P.M. Thompson and J.B. Orlin, The theory of cyclic transfers, Operations Research Center Working Paper, MIT, August 1989.
- [28] P.M. Thompson and H.N. Psaraftis, Cyclic transfer algorithms for multivehicle routing and scheduling problems, *Operations Research* 41 (1993).
- [29] P. Toth and D. Vigo, The granular tabu search (and its application to the vehicle routing problem), Technical Report, Dipartimento di Elettronica, Informatica e Sistemistica, Universit di Bologna, Italy, 1998 (revised version to appear in *INFORMS Journal on Computing*).