

## Singapore Management University Institutional Knowledge at Singapore Management University

---

Research Collection School Of Information Systems

School of Information Systems

---

9-2004

# A Development Framework for Rapid Metaheuristics Hybridization

Hoong Chuin LAU

Singapore Management University, [hclau@smu.edu.sg](mailto:hclau@smu.edu.sg)


M. K. LIM

W. C. Wan

S. Halim

**DOI:** <https://doi.org/10.1109/CMPSAC.2004.1342859>

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)

 Part of the [Artificial Intelligence and Robotics Commons](#), [Business Commons](#), and the [Operations Research, Systems Engineering and Industrial Engineering Commons](#)

---

### Citation

LAU, Hoong Chuin; LIM, M. K.; Wan, W. C.; and Halim, S.. A Development Framework for Rapid Metaheuristics Hybridization. (2004). *28th Annual International Computer Software and Applications Conference (COMPSAC)*. 362-367. Research Collection School Of Information Systems.

**Available at:** [https://ink.library.smu.edu.sg/sis\\_research/1129](https://ink.library.smu.edu.sg/sis_research/1129)

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [libIR@smu.edu.sg](mailto:libIR@smu.edu.sg).

# A Development Framework for Rapid Meta-heuristics Hybridization

Hoong Chuin LAU, Wee Chong WAN, Min Kwang LIM, Steven HALIM  
*National University of Singapore*  
{lauhc, jwan, limmk, stevenha}@comp.nus.edu.sg

## Abstract

*While meta-heuristics are effective for solving large-scale combinatorial optimization problems, they result from time-consuming trial-and-error algorithm design tailored to specific problems. For this reason, a software tool for rapid prototyping of algorithms would save considerable resources. This paper presents a generic software framework that reduces development time through abstract classes and software reuse, and more importantly, aids design with support of user-defined strategies and hybridization of meta-heuristics. Most interestingly, we propose a novel way of redefining hybridization with the use of the “request and response” metaphor, which form an abstract concept for hybridization. Different hybridization schemes can now be formed with minimal coding, which gives our proposed Meta-heuristics Development Framework its uniqueness. To illustrate the concept, we restrict to two popular meta-heuristics Ants Colony Optimization and Tabu Search, and demonstrate MDF through the implementation of various hybridized models to solve the Traveling Salesman Problem.*

## 1. Introduction

Meta-heuristics have grown to be an important paradigm in solving large-scale combinatorial optimization problems. Following the success of meta-heuristics such as tabu search (TS), simulated annealing (SA), and genetic algorithms (GA), there has been an explosive growth of new techniques in line with natural and biological observations, such as Ant Colony Optimization (ACO) [5], Squeaky Wheel [10] and Particle Swarm [15]. With the numerous and diverse growth of meta-heuristics, we see the potential for advancing the field further if there is provision for algorithm designers to hybridize one technique with another. As expected, each meta-heuristic has its own forte and shortcoming at solving certain subset of

problems and logically leads to hybrid schemes that could exploit the strengths and cover the weaknesses of one technique with its collaborator(s). Results from the literature have shown that such hybrid methods usually out-perform their predecessors, e.g. [3]. Unfortunately, most hybrid methods are mere mergers of algorithms and often specific to the problems being solved. These hybrid models are usually developed individually and independently with little or no reuse of codes. Such excessive efforts of developing from scratch follows by validation of models are very costly and consequently discourage researchers from exploring new schemes. A high-level software framework is needed to address the above-mentioned issue.

There have been proposals for several algorithm frameworks. For instance, a Java-based tabu search framework, OpenTS [9] has a well-defined structure and an object-oriented programming (OOP) style inherent in the Java language. The design allows the definition of basic elements common to tabu searches through interfaces and performs iterations based on these elements. OpenTS, however, supports only tabu search. The abstract classes also lack the capability to interact with one another and thus may not be easily extended to support hybridization. Gaspero and Schaerf [7] proposed a more general local search framework, EASYLOCAL++, which comprises of meta-heuristics such as tabu search, simulated annealing and composite search. EASYLOCAL++ composes a set of cooperating classes that handle the different aspects of local search. However these classes support only a pre-defined set of hybridization models and do not easily accommodate new schemes. A more established framework is HOTFRAME by Fink and Voß [6] that implements various meta-heuristics such as tabu search, simulated annealing and evolutionary algorithms. HOTFRAME uses template classes to provide reusable data structures on common solution representations as well as their neighborhood operators. The search procedure is performed through

these cooperating classes and hybridization can be achieved through inheritance by overriding the required procedures. However, HOTFRAME could not readily combine two developed applications and thus poses some limitations to the recycling of available systems. Lau et al. [11] recently proposed a Tabu Search Framework (TSF) that uses a centralized concept of handling strategies in a single control mechanism while allowing the flexibility of implementation through abstracts classes. However, TSF offers only tabu search and hybridization while viable, required a major work from the designer.

Localizer++ [14] is an extensible constraint library for local search. It supports both declarative abstractions to describe the neighborhoods as well as high-level search constructs to specify local moves and search procedure. Localizer++ supports a variety of features typically found only in modeling languages and its extensibility allows for an easy integration of new, user-defined, abstractions. While providing ease of configuration, a user must provide Localizer the formulation, albeit of any local search, to construct an algorithm. This differs from the works aforementioned which provides a constructed framework, whereby the usage is simpler since the user simply needs to implement algorithm specific interfaces/classes.

To foster hybridization among algorithms, a higher-level development framework is needed that provides the linkage among implementations of disparate algorithms, thereby enabling easy creation of hybrids. To achieve this purpose, the framework should introduce genericity through promoting reuse of existing techniques and implementations, and yet have the robustness to incorporate implementation of new techniques. Beside the obvious advantage of timesaving in code implementation, the framework allows the algorithm designer to concentrate on algorithmic research and experimentation. In short, we view such a framework to be a foundation tool in advancing meta-heuristics research and development. In this paper, we propose the Meta-heuristics Development Framework (MDF), which extends the work of [11] by working on a higher level where TSF serves as a component algorithm, along with other algorithmic frameworks. With MDF, an algorithm designer can:

- 1) Easily create various hybrid schemes of any existing technique in the framework, or allow others to adapt their algorithm through reuse.
- 2) Benchmark fairly the performance of implementations against an existing technique.

To present the MDF idea, this paper will focus on two different meta-heuristic paradigms, namely TS [8] and ACO [5, 18]. We illustrate the usage of MDF by examining various schemes of hybridization between TS and ACO to solve the Traveling Salesman Problem (TSP). We then compare the additional development time required for each hybrid relative to the pure schemes, with the improvement yield. The rest of the paper is organized as follows. Section 2 presents the architecture of MDF in bringing together the generalized abstract classes common to most meta-heuristics. We also present the hybridization control mechanism using the “request and response” metaphor. In section 3, we present implementation of MDF on six (6) different hybrid models for TSP. Section 4 presents the experimental results and Section 5 the conclusion.

## 2. MDF Architecture

MDF uses abstraction and inheritance as the primary mechanism to build adaptable components or interfaces. The general behavior of local search is abstracted into general interfaces such as *Solution*, *Objective Function*, *Penalty Function*, *Move*, *Constraint* and *Neighborhood Generator*. These interfaces do not deal with the actual algorithm, but provides a common medium, in which different algorithms share information and collaborate. An example can be illustrated through the *Move* interface. In TS, a move is defined as a translation from current solution to its neighbor. For ACO, a move is defined as a transition in the construction from a partial to a complete solution. GA treats a move as a mutation while SA defines the move as a probabilistic operation to its next state. Although each of these operators exhibits a different behavior, their underlying algorithmic concept is the same; they “move” the current solution to its next state.

Proprietary or extended interfaces are built above the generalized interfaces to define unique behaviors exhibited by each meta-heuristic. For example, in ACO, the proprietary interfaces are the *local heuristic* and *pheromone trail*. In TS, these are *tabu list* and *aspiration criteria*. GA has *population* and *recombination* and SA has *annealing function* as their proprietary interfaces. MDF then uses an *Engine* interface for coordinating the rudimentary routines of meta-heuristics through the general and respective proprietary interfaces. Like engine in reality, a *Switch Box* is incorporated to contain the set of tunable parameters for the engine operations. MDF adopts a centralized control mechanism that adapts the search trajectory through user-defined *requests* (events) and

responses (handlers). Hence the search process becomes a request-driven search, in which the occurrences experienced during the search could be utilized to guide future exploration. The architecture of MDF is shown in Figure 1.

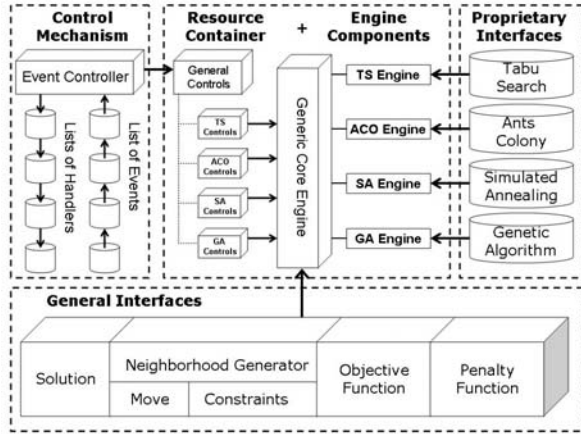


Figure 1: Architecture of MDF.

## 2.1. Hybridization

Hybridization of meta-heuristics is a powerful search strategy that uses the concepts of *Intensification* and *Diversification* [4]. There are two important aspects that must be considered for each hybridized scheme; first is the *point in time* in which certain event(s) occurred, and second is the *required action(s)* to be performed at that time. We define the first aspect as *Requests* and the second aspect as *Responses*. Using this metaphor, the *EventController* class within MDF is the control mechanism for determining the execution of handlers triggered by events.

A formal model for specifying the behavior of the *EventController* is given by a 4-tuple  $\langle S, E, R, s_0 \rangle$ , where:

$S$  is a set of search states  
 $E$  is a set of event/handler pairs  
 $R \subseteq S \times E \times S$  is the transition relation,  
i.e.  $(s, e, s') \in R$  iff  $s \xrightarrow{e} s'$ .  
 $s_0$  is a state in  $S$  denoting the initial search state.

In our model, a search state at any time point comprises the current and best-found solutions, the operating engine and the search parameters. A search algorithm begins with an initial search state. As the search proceeds, event(s) (such as non-improving solutions and new best solution) defined by the algorithm designer will determine the corresponding response (such as switching to another meta-heuristic engine) to be executed. Each request is associated with two parameters: a list of “to-be-executed” responses

and their priority. The hierarchical nature of priority for the responses will allow designers to have additional control over their execution sequences.

The *EventController* is responsible for switching the meta-heuristic engines on their turns to modify the search state. We illustrate this idea using the hybrid scheme of [18] where ACO and Local Search (LS) were used to solve TSP. The generalized idea was to use ACO as the core algorithm and apply LS to improve on the iteration-best solution before the ants updated it into the pheromone trails. This strategy can be implemented in MDF as follows; ACO is the operating meta-heuristic and LS is embedded into a handler (response). We define an event (request) that will be triggered when an iteration-best solution is found during the ACO search. The procedure begins with ACO sending out its ants to find solutions. At the end of iteration, the iteration-best solution found will trigger the LS handler. The *EventController* then hands the search state to LS engine, which in turn improves on the iteration-best solution. Once completed, the *EventController* returns the enhanced solution to ACO engine for pheromone trails update. This process continues until a terminating criterion is reached.

## 3. Hybridization Schemes for TSP

We now show how MDF can be applied to implement the hybridization schemes as proposed by Lau et al. [12]. The authors introduce *Hybrid Ants System Tabu Search (HASTS)* which has four different derived models, namely Empowered Ants (HASTS-EA), Intensification Exploitation (HASTS-IE), Enhanced Diversification (HASTS-ED) and Collaborative Coalition (HASTS-CC). Our objective is to discover the effectiveness of each hybridized schemes on TSP as well as the additional effort required to build them from their pure schemes. In addition, we also implement hyper-hybrids, which combine two hybridized schemes. We found that these hyper-hybrids improve the quality of result for most cases and the additional development cost is relatively trivial. This unlocks a potential area for creating complex hybrids rapidly, yielding improvement to previous results.

### 3.1. Strict Tabu Search (Pure TS)

There are currently various schemes for TS, such as strict TS [8], reactive TS [2] and robust TS [19] and for this illustration, we developed the strict TS. We represent a tour or a *Solution* as a single dimension array and the *Move* is a swap-edge operator that

exchanges two arbitrary edges in the tour. The function of the *Neighborhood Generator* is then to generate the list of possible swaps between the edges. The *Objective Function* is simply to sum up the total distance traveled. Two proprietary interfaces are required to elicit the essence of tabu search; *Tabu List* and *Aspiration Criteria*. In our case, we tabu the swapped edges and apply a static tenure that is equal to the instance size. The aspiration criterion is to override the tabu status of a move if a better objective value is found (see [8]).

### 3.2. Ants Colony Optimization (Pure AC)

We select ACO as our next hybridization candidate. The algorithm is implemented with the settings as proposed in [18]. An interesting observation is that the development time is much less than expected due to the code reuse from the strict TS. Solution and Objective Function for example, has the same formulation in both implementation, and hence can be reused without any modification. In addition, both ACO and strict TS use the same Solution interface and this allows both engines to operate on the generated solutions without further alteration. The Move operator is used to incrementally add a new city to the solution (which is initially empty) and the Neighborhood Generator maintain a list of unvisited cities. A solution is completely constructed when the Neighborhood Generator could not construct new moves (i.e. the unvisited list is empty). We are also required to implement the ACO proprietary interfaces. The Local Heuristic is function that computes the inverse of length of the tour. The Pheromone Trail records on the preferences of ants in following a route when visiting the cities.

### 3.3. HASTS-EA (Empowered Ants)

This hybrid scheme is inspired from observing that ACO cycle near optimal solutions due to the emphasis on the strong pheromone trails. By empowering the ants with memory (tabu list), it reduces the chances of reconstructing the same solution. In short, ACO optimizes the solution based on its pheromone trails as a “preference” memory and reduces solution cycling via the tabu list. Furthermore, tabu search can be applied to modify the solutions radically, hence encouraging exploration that helps to escape from local optimality. The tabu list also eliminates the need for local pheromone decay, which reduces one of the parameters. To implement this strategy, the Neighborhood Generator is modified to include a tabu

list as a handler, which records the solution made by each ant in a single iteration. Subsequent ants in the iteration will trigger an event to check with the handler to prevent reconstruction of similar solutions.

### 3.4. HASTS-IE (Improved Exploitation)

This model is similar to ACO and LS hybrid [18], except that TS is used in the place of LS. The function of the embedded TS is to conduct intensification search on each iteration best solution. This is achieved by using TS to remove the crossings in the solution found by ACO before updating the solution into the pheromone trail. In addition, the memory in TS prevents solution cycling, which results in a more superior search than naïve local search. Consequently, this improved tour will increase the probability of finding a better solution by subsequent ants. In our implementation, TS is applied adaptively by adjusting the terminating criterion with respect to the number of non-improving moves. An event is set to detect the time when an iteration best solution is found. Before the solution is updated into the pheromone trail, a handler will apply TS to optimize the solution until it reaches 100 non-improving moves.

### 3.5. HASTS-ED (Enhanced Diversification)

As TS suffers from local optimality, a diversification strategy is to incorporate a diversifier (e.g. [13]). This hybrid scheme uses ACO as a diversifier for TS. Although there are many diversification schemes such as random restart and probabilistic diversification [17], the diversified solution is often poor. ACO provides a remedy by reconstructing quality solutions. In our implementation, we implement a counter event to adaptively apply ants to diversify as a non-linear function of non-improving moves. A recommended function is to cumulatively increment the number of non-improved move tolerated for every diversification applied. The diversification technique is embedded into the handler, which reconstructs the part of best-found solution in TS using ACO.

### 3.6. HASTS-CC (Collaborative Coalition)

This scheme is a 2-phase approach between ACO and TS, which offers the least coupling between the two meta-heuristics. This scheme is inspired from the observation that ACO works extremely well for the constructing phase while TS is more suited for optimizing the generated solutions. Such collaboration exploits the natural heritage of each meta-heuristic.

This scheme is easily implemented by setting an event to switch from ACO to TS when ACO has completed its iterations.

### 3.7. Two Hyper Hybrid Models

In addition to the four hybrid schemes, we illustrate the strengths of MDF in combining hybrid to form hyper-hybrid. We introduce two hyper hybrid schemes, HASTS-CCED and HASTS-IEEA. HASTS-CCED replaces the TS in HASTS-CC to HASTS-ED. This aims to enhance the optimizing phase. For HASTS-IEEA, it fuses the tabu list strategy in HASTS-EA to HASTS-IE, thus allowing HASTS-IE to develop a more aggressive diversifying capability. HASTS-CCED and HASTS-IEEA illustrates how hyper-hybrids can be easily formed from previously constructed hybrids when MDF is used. Initial experimentation of has shown promising results for these hyper hybrids with low additional development cost.

## 4. Discussion of Results

We demonstrate experimentally the cost-effectiveness of MDF in hybridization using the TSP test problems obtained from TSPLIB [16]. The most obvious and necessary incentive for using a framework is cost-savings in development time. However, it is difficult to measure accurately the amount of efforts required due to the numerous affecting factors. As such, we infer from the lines of codes to reveals at least partially the programming efforts. We understand that judging by the number of line of codes alone is often inadequate to reflect the development time, as some programmers are known to write condensed codes. In addition, this metric only considers the implementation time and not the validation time. Intuitively, if each hybrid scheme was developed independently, they would be validated separately. An implicit advantage of MDF is its capability to recycle validated applications and hence it could save considerable resources especially in complex software such as meta-heuristic hybridization. Unfortunately, this validating cost as well as other development costs could not be easily recorded and we present in Figure 2 the lines of development codes as an informal representation of the developmental efforts. From the comparison, it is apparent that developing strict TS and ACO requires less effort than to build from scratch. The large amount of codes in MDF and the relatively smaller additional codes to formulate TSP, strongly suggests that MDF provides the bulk of the

implementation. Consequently, this implies that MDF has a strong software reuse capability that could greatly abate development time, satisfying the primarily motive of the framework.

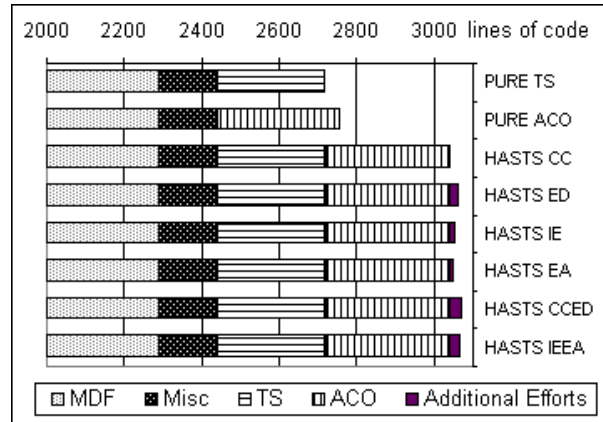


Figure 2: Comparison of development costs

To compare the effectiveness, all test cases are run for different hybrids on an Athlon XP 3200+ processor with 512MB of memory, and the results are taken after 90 seconds, independent of the instance size. We examine the result of KROA150 as an example.

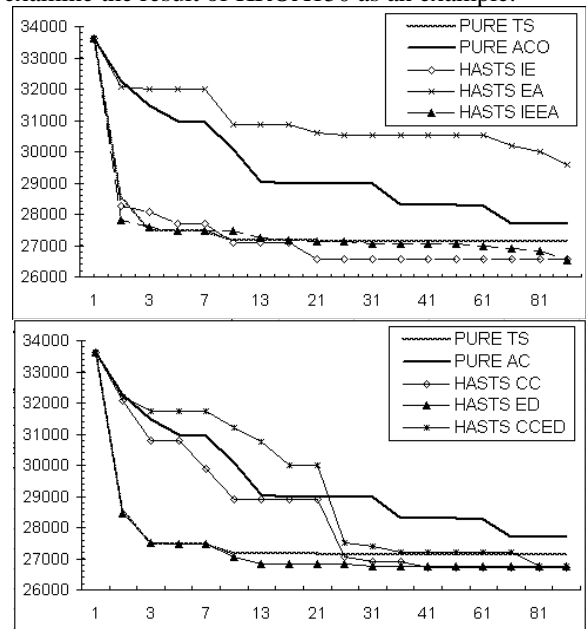


Figure 3: Result of test case KROA150.

We observed that Pure TS converged faster than Pure AC. However the solution quality of TS stops improving at around 10 seconds while Pure AC continued to improve on its solution. HASTS-CC, HASTS-ED and HASTS-CCED produced the same result at 90 seconds although HASTS-ED converged

the fastest. Although HASTS-CCED appeared to be slowest to reach the local optimum, we observe a rapid improvement from 22<sup>nd</sup> sec to the 26<sup>th</sup> sec. The winner of this instance is HASTS-IEEA where the local optimum is reached at 88 seconds. HASTS-EA has the weakest result showing the unsuitability of the scheme in this instance. An addition of another 15 test cases from the TSPLIB is recorded in Figure 4. The “Bound” column shows the best-published results to date. Each column gives the percentage gap as benchmarked against the best-published results. In summary, the table shows that HASTS-IEEA produces the best results and has the best standard deviation. Although it is not conclusive, we have a strong belief that hybrids (and hyper-hybrids) usually out-perform their parents. With MDF, complex hybridized schemes are now possible to be developed in much less development time, allowing complex hybridization to become a practical solution for algorithm improvement.

Name	Bound	A	B	C	D	E	F	G	H
att48	10628	1.19	2.06	0.24	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	2.18	<b>0.00</b>
eil51	426	0.23	0.94	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	0.23	0.94	<b>0.00</b>
pr76	108159	0.95	3.55	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	3.03	<b>0.00</b>
kroA100	21282	0.07	1.30	<b>0.00</b>	<b>0.00</b>	0.05	<b>0.00</b>	3.81	<b>0.00</b>
kroB100	22141	0.42	4.53	0.27	0.31	0.59	0.36	3.59	<b>0.00</b>
eil101	629	<b>0.00</b>	3.18	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	1.43	<b>0.00</b>
ch130	6110	1.41	6.25	0.65	0.29	0.05	0.23	6.25	<b>0.05</b>
kroA150	26524	2.27	4.37	0.77	0.92	0.90	0.10	4.14	<b>0.00</b>
kroB150	26130	0.18	6.81	2.79	0.08	1.00	0.01	9.07	<b>0.00</b>
d198	15780	0.82	10.25	0.10	0.61	0.12	<b>0.00</b>	9.08	0.01
kroA200	29368	0.41	16.07	0.41	1.02	0.80	0.67	22.10	<b>0.38</b>
kroB200	29437	2.32	25.62	2.32	2.32	2.32	1.28	25.62	<b>0.36</b>
a280	2579	3.49	22.41	3.49	3.06	2.91	0.74	22.41	<b>0.00</b>
lin318	42029	2.60	24.10	2.60	2.16	2.51	1.78	19.09	<b>1.51</b>
pcb442	50778	2.46	22.06	2.46	2.13	2.32	2.16	22.06	<b>1.73</b>
<b>Average Gap</b>		1.25	10.23	1.07	0.86	0.90	0.50	10.32	<b>0.27</b>
<b>STD Deviation</b>		1.11	9.15	1.26	1.05	1.07	0.70	9.13	<b>0.56</b>

Figure 4: Tabulation of TSP results.

## 5. Conclusion

MDF speeds up the development of new and hybridized meta-heuristics. It also provides a level-playing field for experimental benchmarking of algorithms. In future, MDF will be enhanced with higher level of genericity and robustness.

## 6. References

[1] Ahuja, R. K., Jha K. C., Orlin J. B. and Sharma D., Very Large-Scale Neighborhood Search for the Quadratic Assignment Problem, MIT Sloan School of Management Working Paper, 2003.

[2] Battiti, R. and Tecchiolli, G. 1994. The reactive tabu search. *ORSA Journal on Computing*, 6:2, 126-140.

[3] Bent, R. and Van Hentenryck, P. 2001. A two stage hybrid local search for the vehicle routing problem with time windows, Technical Report, CS-01-06, Dept. of Computer Science, Brown University.

[4] Blum, C. and Roli, A. 2003. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Survey*, 35:3, 268-308.

[5] Dorigo, M. and Di Caro, G. 1999. *The Ant Colony Optimization Meta-Heuristic*. Readings, New Ideas in Optimization: McGraw-Hill, 11-32.

[6] Fink, A. and Voß, S. 2002. HotFrame: A Heuristic Optimization Framework. Readings, In *Optimization Software Class Libraries*: Kluwer, Boston, 81-154.

[7] Di Gaspero, L. and Schaerf, A. 2001. EASYLOCAL++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. *In Proceedings of the 4th Metaheuristics International Conference*.

[8] Glover, F. and Laguna, M. 1997. *Tabu Search*. Readings, Tabu Search: Kluwer Academic Publishers.

[9] Harder, R. 2001. IBM OpenTS website: <http://opents.iharder.net>

[10] Joslin, D. E. and Clements, D. P. 1999. Squeaky wheel optimization. *In Proceedings of the 15th American Association for Artificial Intelligence*.

[11] Lau, H. C., Wan, W. C., and Jia, X. 2003. Generic Object-Oriented Tabu Search Framework. *In Proceedings of the 5th Metaheuristics International Conference*.

[12] Lau, H. C., Lim, M. K., Wan, W. C., Wang, H and Wu, X. 2003. Solving Multi-Objective Multi-Constrained Optimization Problems using Hybrid Ants System and Tabu Search. *In Proceedings of the 5th Metaheuristics International Conference*.

[13] Li, H. and Lim, A. 2001. A Metaheuristic for the Pickup and Delivery Problem with Time Windows. *In Proceedings of International Conference on Tools with Artificial Intelligence*.

[14] Michel, L. and Van Hentenryck, P. 2001. Localizer++: An Open Library for Local Search, Technical Report, CS-01-03, Brown University

[15] Parsopoulos, K. E. and Vrahatis, M. N. 2002. Particle Swarm Optimization Method for Constrained Optimization Problems. Readings, *Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies*: IOS Press, 214-220.

[16] Reinelt, G. 1991. TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing* 3, 376-384

[17] Rochat, Y. and Taillard, E. 1995. Probabilistic Diversification And Intensification In Local Search For Vehicle Routing. *Journal of Heuristics*, 1, 147-167.

[18] Stützle, T. and Dorigo, M. 1999. ACO Algorithms for the Traveling Salesman Problem. Readings, *Evolutionary Algorithms in Engineering and Computer Science*, Wiley.

[19] Taillard, E. 1991. Robust Tabu Search for the Quadratic Assignment Problem. *Parallel Computing*, 17, 443-455.