Incremental Local Search in Ant Colony Optimization: Why It Fails for the Quadratic Assignment Problem

Prasanna Balaprakash, Mauro Birattari, Thomas Stützle, and Marco Dorigo

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
{pbalapra, mbiro, stuetzle, mdorigo}@ulb.ac.be

Abstract. Ant colony optimization algorithms are currently among the best performing algorithms for the quadratic assignment problem. These algorithms contain two main search procedures: solution construction by artificial ants and local search to improve the solutions constructed by the ants. Incremental local search is an approach that consists in re-optimizing partial solutions by a local search algorithm at regular intervals while constructing a complete solution. In this paper, we investigate the impact of adopting incremental local search in ant colony optimization to solve the quadratic assignment problem. Notwithstanding the promising results of incremental local search reported in the literature in a different context, the computational results of our new ACO algorithm are rather negative. We provide an empirical analysis that explains this failure.

1 Introduction

Ant colony optimization (ACO) is a recent metaheuristic technique that is inspired by the pheromone trail laying and following behavior of some ant species [1]. In ACO algorithms, artificial ants are stochastic solution construction procedures that generate solutions using artificial pheromones and heuristic information; the ants' solutions are then used to modify the artificial pheromone trails. This mechanism shifts the stochastic solution construction procedure towards the construction of solutions similar to the better ones seen previously in the algorithm. The definition of the ACO metaheuristic includes also the possibility of using local search [1]: Once ants complete their solution construction phase, local search algorithms can be used to refine their solutions before using them for the pheromone update. Various experimental researches have shown that the combination of solution construction by ants and local search procedures is a promising approach [1].

There exist a large number of possible choices when using local search in ACO algorithms. We refer the reader to [1,2] for a recent review of these techniques. The primary goal of this paper is to investigate the opportunity of adopting incremental local search in ACO, that is, to improve via a local search algorithm the ants' partial solutions at regular intervals during the solution construction

M. Dorigo et al. (Eds.): ANTS 2006, LNCS 4150, pp. 156–166, 2006.

process. Previous works on incremental local search in non-ACO algorithms have reported promising results: Russel [3] introduced a method for re-optimizing partial solutions by means of an interchange procedure after every k steps of the solution construction. Gendreau et al. [4] introduced a generalized insertion heuristic to solve the traveling salesman problem and extended the same approach to a vehicle routing problem [5]. In a nutshell, generalized insertion can be described as an insertion procedure that uses a limited form of incremental local search. Caseau and Laburthe [6] introduced an approach that applies local search after each step of the solution construction process to solve a large constrained vehicle routing problem. This methodology is compared with a customary technique that constructs solutions by greedy insertion and uses local search at the end to improve solutions. The computational results showed that, in this particular context, incremental local search was not only faster, but also produced much better solutions. They conclude that incremental local search is able to perform some improvements during the construction process that full local search may not be able to perform once the solution is complete. In the context of constructive methods, Fleurent and Glover [7] proposed a strategy called proximate optimality principle that consists in re-optimizing partial solutions of a greedy randomized adaptive search procedure to solve the quadratic assignment problem. They suggest that imperfections introduced during the construction step of the procedure can be removed by applying local search on the partial solutions. Since the method we investigate here is very similar to this experimental study, we have also chosen the quadratic assignment problem for our analysis.

The main motivation behind our research is that *a priori* the idea of re-optimizing the partial solutions of the ants during the solution construction looks promising, since the use of local search in ACO algorithms has already proven to often lead to a strong improvement of performance, and incremental local search has been successfully applied in other settings where constructive methods were used. However, the results of our computational experiments are negative and, at least for the quadratic assignment problem (QAP), the inclusion of incremental local search actually worsens the performance. In this paper, we analyse the possible reasons for this effect by studying the convergence behavior of the ACO algorithm. In fact, our analysis also gives hints on conditions under which incremental local search may become useful in ACO algorithms. For instance, since the empirical analysis shows that the incremental local search introduces a strong exploration in the search process of the ACO algorithm studied here, one might try to use it to generate new solutions when the search stagnates.

The paper is organized as follows. Section 2 shows how to use incremental local search in ACO for solving the quadratic assignment problem. In Section 3, we report our computational results, which show that incremental local search in ACO obtains rather poor results. An analysis of why incremental local search in ACO is not effective for the quadratic assignment problem is presented in Section 4. Section 5 concludes the paper.

2 Incremental Local Search in Ant Colony Optimization for the Quadratic Assignment Problem

The QAP can be described in the following way: Consider a set of n facilities that have to be assigned to n locations. A matrix $\mathcal{A} = [a_{rs}]$ gives the distances between locations, where a_{rs} is the distance between locations r and s. A matrix $\mathcal{B} = [b_{tu}]$ characterizes the flows among facilities, where b_{tu} is the flow between facility t and facility u. An assignment can be represented by a permutation π of $\{1,\cdots,n\}$, where $\pi(r)$ is the facility that is assigned to location r. The problem is to find a permutation π^* that minimizes the sum of the products of the flows among facilities by the distance between their locations.

Among the various metaheuristics, ACO has been shown to be particularly successful on the QAP as best exemplified by the high performance reached by $\mathcal{MAX}-\mathcal{MIN}$ ant system ($\mathcal{MM}$AS-QAP) [8,9]. Hence, we have chosen $\mathcal{MM}$AS-QAP as a starting point for our analysis. $\mathcal{MM}$AS-QAP constructs solutions by assigning at each construction step a facility to some location. Pheromone trails τ_{ij} refer to the desirability of assigning facility j to location i and the usual probabilistic choice known from ant system is used; since $\mathcal{MM}$AS-QAP does not use any form of heuristic information, the probability p_{ij} of assigning facility j to location i is directly proportional to τ_{ij} for feasible assignments. The pheromone update is done by lowering the pheromone trails by a constant factor ρ and depositing pheromone on the individual solution components of either the best solution in the current iteration (*iteration-best*), the best solution found so far by the algorithm (*best-so-far*), or the best solution found since the last re-initialization (*restart-best*) of the pheromone trails. We refer the reader to [9] for a more detailed description of $\mathcal{MM}$AS-QAP.

$\mathcal{MM}$AS-QAP uses an iterative improvement algorithm based on the 2-exchange neighborhood, where the set of neighbors of a permutation π comprises all permutations that can be obtained by exchanging the location of two facilities. This iterative improvement algorithm is referred to as 2-opt. When using 2-opt in $\mathcal{MM}$AS-QAP, each ant constructs a feasible solution and improves it by this local search.

It is straightforward to include incremental local search in $\mathcal{MM}$AS-QAP. While in the original $\mathcal{MM}$AS-QAP the local search is applied only to complete solutions, in a version that uses incremental local search, the local search is performed on an ants' partial solution. For convenience, let us define some terminology: We denote for each ant the *number of local searches* applied to its (partial) solutions by i, where i, $1 < i \leq n$, is a user defined parameter. We call $\mathcal{MM}$AS-QAP with incremental local search as $\mathcal{MM}$AS-QAP(i). For example, $\mathcal{MM}$AS-QAP(2) refers to the $\mathcal{MM}$AS-QAP algorithm with an incremental local search in which for each of the m ants the (partial) solution is re-optimized **twice**; for $\mathcal{MM}$AS-QAP(3) three local searches are applied. For the sake of uniformity, we denote the original $\mathcal{MM}$AS-QAP algorithm in which a single local search is performed at the end of the solution construction by $\mathcal{MM}$AS-QAP(1). We use the convention that the local searches are applied after equal sized intervals in the solution construction. Let k be the *number of assignments*;

then, in $\mathcal{MM}$AS-QAP(2) local search is applied after $k = 1 \cdot \lfloor n/2 \rfloor$ assignments and on the complete solutions, while in $\mathcal{MM}$AS-QAP(3), local search is applied after $k = 1 \cdot \lfloor n/3 \rfloor$ and $k = 2 \cdot \lfloor n/3 \rfloor$ assignments are done, and again once the assignment is completed.

In the local search on partial solutions, the cost difference for exchanging two facilities s and r in the partial solution is obtained by the instance data restricted to the occupied locations and used facilities of the current partial solution. The current partial solution is replaced if the local search finds a better neighboring one, and the local search continues until there is no more improvement. From this locally optimized partial solution, the particular ant continues its solution construction process. We expect that the computation time for $\mathcal{MM}$AS-QAP(i) increases with i, since more local searches need to be applied. However, each local search applies to a smaller instance and the final local search on the complete solution may start from an already improved solution, hence, requiring less improvement steps; this counteracts the effect on the computation time incurred by the increased number of local searches. Thus, the rate of increase in the computation time per solution is expected to be less than the rate of increase of i.

3 Experiments

We studied the impact of incremental local search in $\mathcal{MM}$AS-QAP on ten instances from QAPLIB [10] ranging in size from $n = 60$ to $n = 150$. The tested instances fall into one of the following groups: (i) instances with the distance and flow matrix entries generated randomly according to a uniform distribution, (ii) instances whose distance matrix is defined as Manhattan distance between points on a grid, and (iii) randomly generated instances in which the matrix entries are similar to those of real-life QAP instances. We allowed 10 independent trials for each algorithm and the code was run on a dual AMD Opteron$^{\text{TM}}$244 1.75GHz processor, 2 GB RAM and 1 MB L2-Cache. The parameter values for $\mathcal{MM}$AS-QAP are set as proposed in [9] except that the value of ρ is set to 0.1 which results in slightly better performance than the setting $\rho = 0.8$ proposed in the literature. (We run additional experiments that verified that the conclusions drawn in the following do not depend on the parameter value for ρ.) For $\mathcal{MM}$AS-QAP(i), we vary the value of i from 1 to 10 and report the solution quality obtained as the percentage deviation from the best known solutions.

For each instance, we first run $\mathcal{MM}$AS-QAP(1) for 1000 iterations and measured the average time over 10 trails. This average time is then taken as the termination criterion for all algorithms to ensure that we compare the algorithms using a same computation time. Table 1 shows the average solution cost for all values of i as the percentage deviation from the best known solution.

From Table 1, we can observe that the average solution cost obtained by $\mathcal{MM}$AS-QAP(i), for $i \geq 2$ is worse than that of $\mathcal{MM}$AS-QAP(1) for all instances; the only exception is that $\mathcal{MM}$AS-QAP(2) is better than $\mathcal{MM}$AS-QAP(1) on instance `tai150b`. In Table 2, we give the average number of iterations that each of the algorithm variants was able to do in the computation time

Table 1. Experimental results of $\mathcal{MM}$AS-QAP(i) algorithms on several QAP instances; given is, for each instance, the average percentage deviation from the best known solution. All algorithms were stopped after the same computation time. Best results are in bold-face.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
tai60b	**0.0004**	0.0086	0.0335	0.0449	0.0610	0.0557	0.0428	0.0578	0.0694	0.0690
tai80a	**1.2304**	1.6836	2.0250	2.1809	2.3091	2.2281	2.2934	2.3209	2.3216	2.3115
tai80b	**0.0204**	0.0535	0.1294	0.0806	0.0932	0.1052	0.1110	0.1107	0.1176	0.1207
sko81	**0.0672**	0.1433	0.2556	0.2714	0.2786	0.3114	0.3215	0.3125	0.3415	0.3331
sko90	**0.1376**	0.2089	0.2382	0.2863	0.3469	0.3477	0.3837	0.3436	0.4042	0.4168
sko100a	**0.1222**	0.1888	0.2193	0.2826	0.3342	0.3517	0.3536	0.3530	0.3505	0.4477
tai100a	**0.3579**	0.8090	0.9702	1.2808	1.2440	1.1532	1.1214	1.0984	1.0902	1.1808
tai100b	**0.0452**	0.1009	0.1289	0.1655	0.2456	0.2432	0.2391	0.2540	0.2632	0.2723
tho150	**0.2115**	0.2909	0.3530	0.3527	0.4230	0.4336	0.4677	0.5053	0.5104	0.5344
tai150b	0.2757	**0.1935**	0.2858	0.2852	0.4685	0.5319	0.5184	0.6217	0.6654	0.6968

Table 2. Experimental results of $\mathcal{MM}$AS-QAP(i) algorithms on several QAP instances; given is, for each instance, and algorithm pair, the average number of iterations done in a same computation time

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
tai60b	983.1	837.3	711.2	614.9	544.4	489.3	373.2	352.0	354.9	354.0
tai80a	987.7	788.9	450.2	584.3	517.2	355.1	338.8	390.9	338.9	337.9
tai80b	991.2	782.2	460.3	569.6	503.6	353.7	333.9	376.0	325.4	325.8
sko81	1082.1	696.4	768.8	482.5	444.3	419.4	399.8	361.3	417.7	323.6
sko90	1097.7	909.9	769.1	486.7	591.1	531.7	408.6	363.0	414.6	387.6
sko100a	1070.6	890.1	522.5	653.5	578.3	411.5	376.8	363.9	333.8	380.7
tai100a	909.1	757.0	426.6	561.5	501.2	349.3	318.6	309.5	279.2	327.9
tai100b	981.7	760.8	455.4	555.5	488.0	344.0	318.2	302.5	280.4	316.2
tho150	998.2	791.2	663.6	416.0	504.7	454.8	329.1	318.9	299.1	330.3
tai150b	979.1	793.1	679.6	465.4	439.0	393.4	286.4	272.3	257.7	284.5

that was determined as described above. As we had conjectured in the previous section, with increasing value of i, generally also the number of iterations done by $\mathcal{MM}$AS-QAP(i) decreases.

Taking into account this latter observation on the number of iterations run, we could tentatively attribute the reason for the worse performance of $\mathcal{MM}$AS-QAP(i) to the fact that it could generate a smaller number of complete solutions in the same time. Naturally, the question arises: what will happen if all the algorithms are allowed to generate the same number of complete solutions? To answer this question, we re-run all the $\mathcal{MM}$AS-QAP(i) allowing each to perform 500 iterations. These results are given in Table 3. As it can easily been seen, the average solution quality of $\mathcal{MM}$AS-QAP(1) for most instances is still better than that of $\mathcal{MM}$AS-QAP(i) for $i \geq 2$. There are only two exceptions: instances tai80b and tai150b. This means that, in general, the usage of

Table 3. Experimental results of $\mathcal{MM}$AS-QAP(i) algorithms on several QAP instances; given is, for each instance, the average percentage deviation from the best known solution. All algorithms were stopped after 500 iterations.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
tai60b	**0.0056**	0.0195	0.0435	0.0649	0.0564	0.0535	0.0583	0.0729	0.0546	0.0546
tai80a	**1.2951**	2.0692	2.0393	2.3263	2.3028	2.1805	2.1112	2.2939	2.2204	2.2204
tai80b	0.1287	0.1128	0.1300	**0.0734**	0.0907	0.0800	0.1495	0.1042	0.1021	0.1021
sko81	**0.1279**	0.1872	0.2463	0.2573	0.2630	0.3391	0.3224	0.3072	0.3828	0.3074
sko90	**0.1391**	0.2669	0.2747	0.2925	0.4189	0.3746	0.3730	0.3768	0.3370	0.4409
sko100a	**0.1736**	0.2115	0.2318	0.3160	0.3307	0.3867	0.3369	0.3849	0.3267	0.3757
tai100a	**0.4320**	0.8002	1.0031	1.2218	1.3383	1.1167	1.0470	1.0566	1.0507	1.1497
tai100b	**0.0748**	0.0903	0.1507	0.1397	0.2198	0.2111	0.2164	0.2397	0.2133	0.2555
tho150	**0.2347**	0.3456	0.3827	0.3633	0.4236	0.4384	0.4236	0.4736	0.4787	0.4615
tai150b	0.3318	**0.2598**	0.4174	0.3835	0.4508	0.4320	0.4678	0.5603	0.4773	0.6855

incremental local search is actually causing a deterioration of $\mathcal{MM}$AS-QAP's performance, although it is allowed much more computation time. Said in other words, incremental local search is not only computationally expensive but also interferes negatively with the solution process of the ACO algorithm–at least for the QAP.

One may stop here and simply report this as a negative result. However, we were wondering as to why there may be a negative influence of incremental local search into the ACO algorithm's search process. A possible answer to this question is given in the next section.

4 Analysis

In this section, we try to explain why the incremental local search produces a detrimental effect on $\mathcal{MM}$AS-QAP's performance. For motivating the main line of attack of this analysis, let us consider first what is known about the convergence behavior of $\mathcal{MM}$AS-QAP, a high-performing ACO algorithms. Essentially, the search process of $\mathcal{MM}$AS-QAP shows a transition from an exploration phase, which is characterized by iteration-best update (the best solution in the current iteration is allowed to update the pheromones) and relatively high branching factor, to an exploitation phase, where the search is directed towards a search space region whose center is defined by the best-so-far solution seen by the algorithm. Interestingly, while in the exploration phase good quality solutions can already be found, typically the highest quality solutions in a trial of $\mathcal{MM}$AS-QAP are found when the search is in its exploitation phase–characterized by a low branching factor and by the fact that the solutions generated by the ants are relatively close to the best-so-far solution. In fact, $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System was designed with the explicit intention to allow a careful transition between the exploration and exploitation phases and to further avoid search stagnation in the exploitation phase [11,8].

We suspected that the local changes introduced by the incremental local search disturb the behavior of $\mathcal{MM}$AS-QAP in the exploitation phase. This suspicion is based on the fact that the local search on the partial solution does not take into account the pheromone trails and, hence, may lead to significant changes to the partial solution. As an effect of this, the final complete solution before the final local search phase may actually be rather far from the best-so-far solution and, thus, hinder the exploitation phase from being effective. This effect is certainly increased for an increased frequency of partial re-optimizations. Differently, if ants do not apply partial re-optimization, the solutions they construct are relatively close to the best-so-far solution.

Hence, we decided to examine more carefully the variability of the generated solutions by the various settings of i in $\mathcal{MM}$AS-QAP(i). The main idea of our analysis is to examine the distance of the complete solutions generated in the current iteration from the best-so-far solution before and after the final application of the local search algorithm on the complete solutions. This is done for all the 10 settings of i. Since $\mathcal{MM}$AS-QAP(1) has proven to be a state-of-the-art ACO algorithm for the QAP, we use the computed distance as a yardstick for the analysis. For our analysis, we compute the distance $d(\pi, \pi')$ between two solutions π and π' as the minimum number of applications of 2–exchange moves that are required to convert one solution into the other one. This distance measure reflects that deviations in the individual assignments from the best-so-far solution can be undone by exchanging facilities between locations. Note that this distance measure can easily be computed using a linear time algorithm [12].

To make our analysis simpler, we consider a variant of $\mathcal{MM}$AS-QAP, denoted as *rb*$\mathcal{MM}$AS-QAP. In this variant, only the *restart-best* solution, the best solution found since the last re-initialization of the pheromones, is allowed to update the pheromones. (Hence, the *best-so-far* and *iteration-best* solutions are not taken into account in the pheromone deposit.) To justify the usage of *rb*$\mathcal{MM}$AS-QAP in the analysis, we first tested whether the versions *rb*$\mathcal{MM}$AS-QAP(i), $i = 1, \ldots, 10$ shows the same type of behavior as $\mathcal{MM}$AS-QAP(i), using as stopping criterion again 500 iterations. Table 4 shows that this is essentially the case, although *rb*$\mathcal{MM}$AS-QAP gives overall slightly worse results than $\mathcal{MM}$AS-QAP. Nevertheless, we can conclude that *rb*$\mathcal{MM}$AS-QAP captures the same trend as $\mathcal{MM}$AS-QAP and that it is safe to limit the analysis to *rb*$\mathcal{MM}$AS-QAP.

In our analysis, we compute at each iteration of *rb*$\mathcal{MM}$AS-QAP(i), $i = 1, \ldots, 10$, the distance between an ants' *complete* solution and the *restart-best* solution before and after the final local search on the complete solution has been applied. These distances are then averaged across the $m = 5$ ants. We denote these two measures as d^- (average distance *before* the final local search) and d^+ (average distance *after* the final local search), respectively. Figure 1 illustrates the observed results for the values of d^- and d^+ obtained by *rb*$\mathcal{MM}$AS-QAP(i), $i = 1, \ldots, 10$, over 500 iterations for the instance `sko100a`. The trend shown in these plots is representative for all the other instances we tested. (We used Tukey's (Running Median) Smoothing [13] for plotting the curves. If no

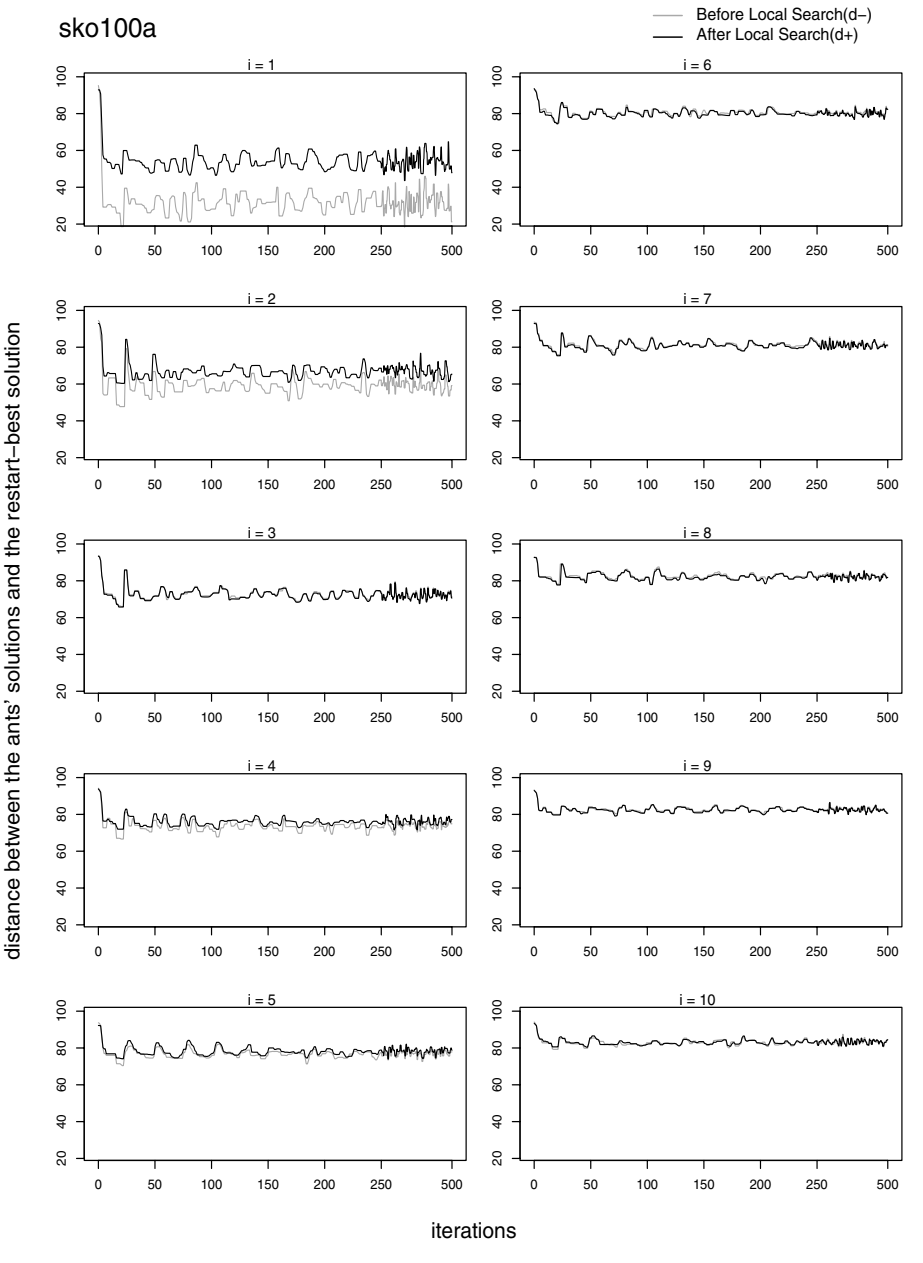

Fig. 1. Experimental results of $rb\mathcal{MM}$AS-QAP on instance `sko100a`. Each plot represents the average distance between the ants' solutions and the *restart-best solution* before and after the final local search on a completed solution for $rb\mathcal{MM}$AS-QAP(i), $i = 1, \ldots, 10$. The stopping criterion is set to 500 iterations.

Table 4. Experimental results of $rb\mathcal{MM}$AS-QAP(i) algorithms on several QAP instances; given is, for each instance, the average percentage deviation from the best known solution. All algorithms were stopped after 500 iterations.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
tai60b	**0.0168**	0.0473	0.0562	0.0620	0.0540	0.0745	0.0643	0.0638	0.0585	0.0585
tai80a	**1.7958**	1.9787	2.1666	2.3629	2.4193	2.1998	2.1897	2.2693	2.2188	2.2188
tai80b	0.1708	**0.0952**	0.0999	0.1062	0.1342	0.0987	0.1069	0.1454	0.1448	0.1448
sko81	**0.1789**	0.2312	0.3063	0.2773	0.3072	0.3303	0.3459	0.3134	0.3749	0.3417
sko90	**0.2288**	0.2835	0.3000	0.3290	0.3664	0.4189	0.3875	0.3720	0.4062	0.4620
sko100a	**0.1982**	0.2830	0.2665	0.3414	0.3768	0.3519	0.3353	0.3942	0.3725	0.4135
tai100a	**0.6073**	0.9753	0.9763	1.3606	1.3204	1.1283	1.0974	1.1031	1.0001	1.1883
tai100b	**0.0938**	0.1231	0.1583	0.2297	0.2368	0.2242	0.2450	0.2811	0.3014	0.3303
tho150	**0.2800**	0.3405	0.4061	0.3990	0.4490	0.4966	0.4540	0.4924	0.4653	0.5229
tai150b	**0.3076**	0.3318	0.3907	0.3681	0.4677	0.5531	0.5978	0.6605	0.5857	0.6691

differences among the curves are visible in the plots, this essentially means that d^- and d^+ are about the same.)

Several important observations can be made from Figure 1. Firstly, the lowest values for d^- and d^+ are reached by $rb\mathcal{MM}$AS-QAP(1). The difference to the other configurations with $i \geq 2$ is smallest for $i = 2$ but then rises quickly with i. (Recall that $rb\mathcal{MM}$AS-QAP(2) performs best among the versions that use incremental local search, as can be seen from Table 1.) Hence, we can conclude that incremental local search on the partial solutions eventually leads to solutions which are very different from *restart-best* solution. Interestingly, for $rb\mathcal{MM}$AS-QAP(1) the values for d^+ are much larger than d^-, which indicates that $rb\mathcal{MM}$AS-QAP(1) can still explore a significant part of the search space, despite the fact that it converges quickly to the exploitation phase, as indicated by the low values of d^-.

Overall, these results confirm our hypothesis that the incremental local search interferes negatively with the exploitation phase of the ACO algorithm and induces a too strong exploration of the search space. For example, for $rb\mathcal{MM}$AS-QAP(1) we have that d^- is around 30 for instance `sko100a`, while for $rb\mathcal{MM}$AS-QAP(2) it increases to about 60—roughly double. Hence, already one incremental local search that is applied after $k = n/2$ assignments have been done, leads to a rather strong perturbation in the exploitation phase, that is, to solutions that are rather distant from the restart-best one. The raise in the values of d^+ is not as strong as for d^-; however, for $rb\mathcal{MM}$AS-QAP(2) d^+ is already significantly larger than for $rb\mathcal{MM}$AS-QAP(1), explaining also $rb\mathcal{MM}$AS-QAP(2)'s worse behavior, in general. As said, these observations also hold for all other instances; detailed results are available from `http://iridia.ulb.ac.be/supp/IridiaSupp2006-002/`.

Finally, we run also experiments for the incremental local search when starting from random initial solutions, to check whether in such an environment the incremental local search can have some contribution. (We have chosen random initial solutions, since for the QAP no high-performing construction heuristics are available.) We run the random restart local search algorithm for the same

Table 5. Experimental results of the random-restart local search on several QAP instances; given is, for each instance, the average percentage deviation from the best known solution. All algorithms were stopped after the same computation time as the algorithms in Table 1. The best results are in bold-face.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
tai60b	**0.1461**	0.1833	0.1991	0.2210	0.1956	0.2202	0.2254	0.2298	0.1716	0.1716
tai80a	2.5402	2.5403	2.4038	2.5176	2.4891	**2.2537**	2.2732	2.2674	2.3156	2.3156
tai80b	0.6667	**0.3876**	0.6568	0.6274	0.6334	0.5372	0.6372	0.5625	0.6668	0.6668
sko81	0.7602	0.7085	0.7545	0.6219	0.6211	0.6041	0.6094	**0.5591**	0.6448	0.6006
sko90	0.8148	0.7860	0.7625	0.7026	0.7263	0.7559	0.7137	**0.6285**	0.6880	0.6734
sko100a	0.7448	0.7449	0.6419	0.6831	0.6356	0.6355	0.5878	0.6528	**0.5590**	0.6494
tai100a	1.4651	1.5655	1.2888	1.3980	1.4050	1.1792	1.1433	1.0761	**1.0518**	1.2299
tai100b	0.6754	0.6708	0.6458	0.5898	0.6056	**0.5688**	0.6294	0.5817	0.5809	0.5914
tho150	0.9266	0.8679	0.8999	0.8185	0.8746	0.8444	0.7996	**0.7739**	0.7825	0.8564
tai150b	1.1589	1.2146	1.2039	1.1491	1.234	1.1430	1.1819	1.1388	1.0972	**1.0522**

average computation time as needed for $\mathcal{MM}$AS-QAP(1) to perform 1000 iterations. Table 5 shows the average solution cost as the percentage deviation from the best known solution, obtained by this random restart local search with different numbers of local searches performed on the (partial) solutions. These results clearly show that for almost all instances, the usage of the incremental local search improves the performance over the version where only once a local search is run on a complete solutions. Hence, these results agree with the computational results reported in the literature [3,4,6,7] and indicate that the usage of incremental local search can be, in some situations, helpful. In fact, random restart has no means to exploit the possibility of learning and exploiting the most promising region of the search space. This suggests that the usefulness of the incremental local search depends strongly on the context where it is applied and the solution construction procedure.

5 Conclusions

Motivated by the promising results of incremental local search reported in the literature [3,4,6,7], we have investigated its behavior and performance in an ACO algorithm for solving the QAP. Our computational study has shown, however, rather poor results for this idea. Next, we have carried out an analysis that can explain this failure. In fact, we have shown that the incremental local search somehow destroys the behavior of the ACO algorithm in its exploitation phase by not allowing it to generate solutions that are rather close to the restart-best or global-best solutions.

Certainly, our results and explanation is limited to the QAP. However, we conjecture that the very same issue arises also in applications of ACO algorithms to other challenging combinatorial problems. More in general, our results also indicate that probably a more careful study of the behavior of ACO algorithms in

the exploitation phase should be done to understand, which techniques may be more promising for improving the performance of ACO algorithms. Finally, our results indicate that incremental local search could be useful for increasing the exploration in convergence situations of ACO algorithms. Although this was not useful on the QAP, it may well be that the careful, occasional addition of incremental local searches in specific situation, could possibly result in improvements for ACO algorithms.

Acknowledgments. This research has been supported by COMP^{2}SYS, a Marie Curie Early Stage Research Training Site funded by the European Community's Sixth Framework Programme under contract number MEST-CT-2004-505079, and by the ANTS project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Thomas Stützle and Marco Dorigo acknowledge support from the Belgian FNRS of which they are a Research Associate and a Research Director, respectively. The information provided is the sole responsibility of the authors and does not reflect the opinion of the sponsors. The European Community is not responsible for any use that might be made of data appearing in this publication.

References

1. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA (2004)
2. Stützle, T., Hoos, H.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
3. Russell, R.: Hybrid Heuristics for the Vehicle Routing Problem with Time Windows. Transportation Science **29** (1995) 156–166
4. Gendreau, M., Hertz, A., Laporte, G.: New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. Operations Research **40**(6) (1992)
5. Gendreau, M., Hertz, A., Laporte, G.: A Tabu Search Heuristic for the Vehicle Routing Problem. Management Science **40** (1994) 1276–1290
6. Caseau, Y., Laburthe, F.: Heuristics for Large Constrained Vehicle Routing Problems. Journal of Heuristics **5**(3) (1999) 281–303
7. Fleurent, C., Glover, F.: Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory. INFORMS Journal on Computing **11**(2) (1999) 198–204
8. Stützle, T., Hoos, H.: $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Future Generation Computer Systems **16**(8) (2000) 889–914
9. Stützle, T., Dorigo, M.: ACO Algorithms for the Quadratic Assignment Problem. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London, UK (1999) 33–50
10. Burkard, R., Karisch, S., Rendl, F.: (`http://www.seas.upenn.edu/qaplib`)
11. Stützle, T., Hoos, H.H.: Improving the Ant System: A Detailed Report on the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Technical Report AIDA–96–12, FG Intellektik, FB Informatik, TU Darmstadt (1996)
12. Schiavinotto, T., Stützle, T.: Metrics on Permutations for Search Space Analysis. Computers & Operations Research (In press)
13. Cohen, P.R.: Empirical Methods for Artificial Intelligence. MIT Press, Cambridge, MA (1995)