Local Probing Applied to Network Routing

Olli Kamarainen[1] and Hani El Sakkout[2]

[1] IC-Parc, Imperial College London, London SW7 2AZ, UK
okl@icparc.ic.ac.uk
[2] Parc Technologies Ltd, Tower Building, 11 York Road, London SE1 7NX, UK
hani@parc-technologies.com

Abstract. Local probing is a framework that integrates (a) local search into (b) backtrack search enhanced with local consistency techniques, by means of probe backtrack search hybridization. Previously, local probing was shown effective at solving generic resource constrained scheduling problems. In this paper, local probing is used to solve a network routing application, where the goal is to route traffic demands over a communication network. The aim of this paper is (1) to demonstrate the wider applicability of local probing, and (2) to explore the impact of certain local probing configuration decisions in more detail. This is accomplished by means of an experimental evaluation on realistic networking scenarios that vary greatly in their characteristics. This paper yields a better understanding of local probing as well as a versatile local probing algorithm for network routing.

1 Introduction

1.1 Local Probing

Due to its systematic nature and support of constraint propagation, *backtrack search enhanced with local consistency techniques* (BT+CS) is effective at solving tightly-constrained problems with complex constraints. On the other hand, the quality of *local search*'s (LS) total assignment is more easily measurable, by comparison with the quality of conventional BT+CS's partial assignments. Also, the absence of systematicity allows LS's assignments to be modified in any order, and so early search moves do not necessarily skew search to focus only on particular sub-spaces. This leads to LS's superiority at optimizing loosely constrained problems. However, while BT+CS algorithms are usually **sat-complete** (i.e. in a finite number of steps, it either generates a solution, or proves that no solutions exist) and can be made **opt-complete** (i.e. in a finite number of steps, it either generates an *optimal* solution, or proves that no solutions exist), LS algorithms are incomplete in both senses.

Local probing [10] is a sat-complete *probe backtrack search* framework (PBT, [6]) that executes a slave LS procedure at the nodes of the master BT+CS search tree. Local probing is designed to tackle practical constraint satisfaction problems (CSPs) and constraint satisfaction and optimization problems (CSOPs) that are difficult solve by pure BT+CS or pure LS algorithms. Its strength is derived from the combination of LS's non-systematic search characteristic with BT+CS's search systematicity. This enables local probing to satisfy complex constraints and prove infeasibility, while achieving good optimization performance, as well as (in some configurations) prove optimality.

In local probing, the constraints of the CSP or the CSOP to be solved must be divided into two sets, namely 'easy' constraints and 'hard' constraints.[1] **The slave LS procedure** — the LS prober — solves sub-problems containing only the 'easy' constraints, at the nodes of a master BT+CS search tree. **The master BT+CS procedure** eliminates possible violations of 'hard' constraints by incrementally posting and backtracking additional 'easy' constraints.

The LS prober algorithm will return a solution to the 'easy' sub-problem. If the LS prober's solution happens to be feasible with 'hard' constraints also, a feasible solution to the entire problem is found. If the LS solution for the 'easy' sub-problem violates any 'hard' constraints, one of them must be selected for repair: A new 'easy' constraint, *forcing the LS prober to avoid returning solutions that violate the constraint in the same way*, is posted to the 'easy' sub-problem. If this leads to a failure (i.e. the new 'easy' sub-problem is unsatisfiable), the negation of the selected 'easy' constraint is imposed instead. Then, the new 'easy' sub-problem is solved with LS, and in the case of further 'hard' constraint violations, other 'easy' constraints are recursively imposed. This continues until a solution is found, or until all possibilities of posting 'easy' constraints are explored (i.e. a proof that no solutions exist is obtained). In the case of a CSOP, the search can be continued again by using a cost bound like in other branch and bound (B&B) methods.

A sat-complete local probing algorithm must satisfy the following conditions:

1. (a) Any 'hard' constraint must be expressible as a disjunction of *sets of 'easy' constraints* that apply to its variables $\langle ES_1, ES_2, \cdots, ES_k \rangle$, such that every solution that satisfies one of the 'easy' constraint sets is guaranteed to also satisfy the 'hard' constraint.
   (b) No solution exists that satisfies the 'hard' constraint, but does not satisfy any of the 'easy' constraint sets in this disjunction.[2]
   For efficiency only, the next condition is also useful:
   (c) No solution to an 'easy' constraint set ES_i in the disjunction is also a solution to another 'easy' constraint set ES_j in the disjunction.
2. If a 'hard' constraint is violated, it must be detected and scheduled for repair by the master BT+CS procedure within a finite number of steps.
3. The master BT+CS procedure must be capable of (eventually) unfolding all possible sets of 'easy' constraints whose satisfiability would guarantee that the 'hard' constraint is also satisfied.
4. It must systematically post 'easy' constraints (and on backtracking, their negations) to the LS prober until either the 'hard' constraint is satisfied, or it is proved impossible to satisfy.

[1] This decomposition should not be confused with hard/soft constraints.
[2] We are theoretically guaranteed to find at least one partition satisfying 1 for any finite CS(O)P (define 'easy' constraints to be variable assignments or their negations).

5. The neighbourhood operator of the LS prober must guarantee to satisfy the posted 'easy' constraints if this is possible (and indicate infeasibility if it is not), by being sat-complete w.r.t. the 'easy' constraints.

Additionally, *opt-completeness* can be achieved if (a) the cost bound constraint generated by a B&B process at the master BT+CS level can be captured as an 'easy' constraint that is satisfied by the LS prober; or (b) the cost bound constraint can be dealt with at the master BT+CS level as a 'hard' constraint, ensuring the cost bound will be satisfied by the search if that is possible (which, in fact, is the case in the local probing algorithm detailed in this paper).[3]

Related work. Local probing belongs to the LS/BT+CS hybridization class where LS is performed in (all or some) search nodes of a BT+CS tree.[4] In most of these hybrids, the task of the LS procedure is to support somehow the master BT+CS. They can be classified further as follows: **A.** Improving BT+CS partial assignments using LS, e.g. [5, 15]. **B.** Enhancing pruning and filtering by LS at BT+CS tree nodes, e.g. [7, 17]. **C.** Selecting variables by LS in a master BT+CS, e.g. [14, 19]. **D.** Directing LS by a master BT+CS, e.g. [3][5]. Local probing has aspects of B, C and D, although D is the key classification, since it is LS that creates assignments, and the task of BT+CS is to modify the sub-problem that LS is solving, directing LS to search regions where good solutions can be found.

In addition to LS/BT+CS hybrids, local probing belongs to the family of PBT hybrid algorithms [1, 2, 6, 12] (related ideas also in [8]). Typically, they use LP or MIP, instead of LS, as the prober method to be hybridized with BT+CS.

1.2 Objectives of the Paper

In earlier work [10], we demonstrated that local probing can be effective at solving a generic scheduling problem. We showed how the local probing hybridization framework can successfully marry the optimization strengths of LS and the constraint satisfaction capabilities of BT+CS, and we learned how it is possible to construct an effective sat-complete local probing hybrid that performs well when compared to alternative algorithms. However, several questions remained unanswered.

Firstly, how readily applicable is local probing in different application domains? In particular, is it possible to build efficient sat-complete local probing hybrids for other applications?

Secondly, problem constrainedness can vary greatly. Can local probing be easily configured to trade-off satisfaction performance against optimization performance by changing the balance of effort between BT+CS and LS? Also, for problem instances with similar levels of constrainedness, it is likely that certain

[3] The local probing algorithm presented in [10] was not opt-complete.

[4] Other LS/BT+CS hybridizations either (a) perform BT+CS and LS serially as "loosely connected", relatively independent algorithms (dozens of published hybrids); or (b) use BT+CS in the neighbourhood operator of LS (including operators of GAs), e.g. [9, 16]. An important sub-class of (b) is "shuffling" hybrids, e.g. [4, 18].

[5] The results presented in [3] are particularly promising to local probing research.

local probing configurations are more effective than others. However, could a single configuration remain competitive over problem instances that vary greatly in constrainedness, or are adaptive configuration mechanisms necessary?

The main objective of this paper is to address these questions by means of a detailed investigation of local probing performance on the selected network routing problem. In the process of doing so, we will establish a new and versatile family of algorithms based on local probing for solving this commercially important problem.

Next, we introduce the application domain. Section 2 details the algorithm to be used in the investigation study of Section 3. Section 4 concludes the paper.

1.3 Application Domain

The network routing problem solved here (NR) involves constraints including capacity, propagation delay, and required demands. In the NR, we have:

- a network containing a set of nodes N and a set of directed links E between them — each link (i,j) from a node i to a node j has a limited bandwidth capacity c_{ij} and a propagation delay d_{ij} that is experienced by traffic passing through it;
- a set of demands K, where each $k \in K$ is defined by:
  - a source node s_k, i.e. where the data is introduced into the network;
  - a destination node t_k, i.e. where the data is required to arrive;
  - a bandwidth q_k, i.e. the bandwidth that must be reserved from every link the demand is routed over; and
  - a maximum propagation delay d_k, i.e. the maximum source-to-destination delay that is allowed for the demand k;
- a subset of the demands $RK \subseteq K$ specifying which demands *must* have paths in any solution.

The aim is to assign paths over the network to demands such that:

1. the total unplaced bandwidth, i.e. the sum of the bandwidth requirements of the demands *not assigned to a path*, is minimized;
2. the following constraints are satisfied:
   - If a demand k belongs to the set of *required demands* RK, it must have a path.
   - Every link in the network (i,j) has sufficient bandwidth to carry the demands that traverse it.
   - For any path P_k assigned to a demand $k \in K$, the sum of the propagation delays of the links in the path does not exceed the maximum propagation delay d_k for k.

Note that, when the set of required demands is exactly the set of all the problem demands, i.e. $RK = K$, the problem is a pure CSP without any optimization dimension: each demand must have a feasible path. On the other hand, if RK is empty, it is always possible to find a trivial solution, because *even an empty set of paths is a solution.*

For each demand k, we use a boolean y_k to denote whether k is routed ($y_k = 1$) or not ($y_k = 0$). Another set of booleans x_{ijk} indicate whether a demand k is routed through a link (i,j) ($x_{ijk} = 1$), or not ($x_{ijk} = 0$). Next, we formally state the problem.

$$\min \sum_{k \in K} (1 - y_k) q_k, \tag{1}$$

s.t.

$$\begin{aligned}
\forall k \in RK : \quad & y_k = 1 && (2)\\
\forall k \in K \setminus RK : \quad & y_k \in \{0,1\} && (3)\\
\forall (i,j) \in E, \forall k \in K : \quad & x_{ijk} \in \{0,1\} && (4)
\end{aligned}$$

$$\begin{aligned}
&\forall k \in K :\\
&\quad y_k = 1 \Leftrightarrow && (5)\\
&\qquad P_k = \langle (n_{k_1}, n_{k_2}), (n_{k_2}, n_{k_3}), \ldots, (n_{k_{l-1}}, n_{k_l}) \rangle,\\
&\qquad \text{where :}\\
&\qquad\quad n_{k_1} = s_k\\
&\qquad\quad n_{k_l} = t_k\\
&\qquad\quad \forall (n_{k_p}, n_{k_q}) \text{ in } P_k : \quad (n_{k_p}, n_{k_q}) \in E\\
&\qquad\quad \text{alldifferent}\{n_{k_1}, n_{k_2}, \ldots, n_{k_l}\}\\
&\quad y_k = 0 \Leftrightarrow P_k = \langle\ \rangle && (6)
\end{aligned}$$

$$\begin{aligned}
&\forall (i,j) \in E, \forall k \in K :\\
&\qquad\qquad x_{ijk} = 1 \quad \Leftrightarrow (i,j) \text{ in } P_k, && (7)\\
&\qquad\qquad x_{ijk} = 0 \quad \Leftrightarrow (i,j) \text{ not in } P_k, && (8)\\
&\forall (i,j) \in E : \quad \textstyle\sum_{k \in K} q_k x_{ijk} \le c_{ij} && (9)\\
&\forall k \in K : \quad \textstyle\sum_{(i,j) \in E} d_{ij} x_{ijk} \le d_k && (10)
\end{aligned}$$

The objective function (1) to be minimized is the sum of the bandwidths of the non-routed demands (i.e. those demands k with $y_k = 0$). The constraints of (2) force demands in the set of required demands RK to have a path, and the remaining demands can either have a path or not (3). The path constraints (5) and (6) force, for any placed demand k, its path P_k to be a connected loop-free sequence of links from the source node s_k to the destination node t_k, and for any non-routed demand, its path to be empty. The constraints in (7) and (8) tie the link-demand booleans x_{ijk} with the paths. The capacity constraints in (9) ensure that for each link (i,j) the bandwidth reserved for demands passing through it does not exceed its capacity c_{ij}. The delay constraints (10) restrict for each demand k the total delay of the links in its path to be no more than its maximum propagation delay d_k.

2 Algorithm Description

The problem decomposition and the master BT+CS of local probing are introduced first. Then the LS prober and its neighbourhood operator are described.

2.1 Problem Constraint Decomposition into 'Easy' and 'Hard'

The NR is a typical large-scale combinatorial optimization problem (LSCO) in that it can be divided into different sub-problems. The constraints of the problem are divided into the 'easy' and 'hard' constraints here as follows:

- 'Hard' constraints (not guaranteed to be satisfied by the prober):
  - Capacity constraints: the sum of the bandwidth requirements of the demands routed via a link must not exceed the *bandwidth of the link* (9)
- 'Easy' constraints (guaranteed to be satisfied by the prober):
  - The sum of the propagation delays of the links in a path is less than or equal to the *maximum propagation delay* of the demand (10)
  - 'Easy' constraints that the master BT+CS procedure can impose during search (but might also exist in the original problem):
    - If the *demand k is dropped* ($y_k = 0$), it cannot have a path (6).
    - If the *demand k is required* ($y_k = 1$), it must have a valid path (5).
    - If a *link is forbidden* ($x_{ijk} = 0$) from a demand, its path is not allowed to contain the forbidden link (8).
    - If a *link is forced* ($x_{ijk} = 1$) for a demand, its path must contain the forced link (7).

2.2 Master BT+CS Level

The last four of the constraints above are used by the master BT+CS procedure to repair violations on 'hard' constraints, i.e. link capacity constraints, in the following way. Assume that the prober returns a probe (a set of paths for all non-dropped demands), and the master BT+CS procedure detects that the bandwidth usage on link (i, j) exceeds its capacity, and then selects this link to be repaired (see Fig. 1). **1.** The algorithm picks one demand k traversing (i, j), and *drops* it, i.e. posts a constraint $y_k = 0$. This first search branch does not subsequently allow a path for the demand k. **2.** If dropping k does not lead to a solution in the subsequent nodes, a constraint that requires that demand k has a path, i.e. $y_k = 1$, and a constraint that forbids the link (i, j) from the demand k, i.e. $x_{ijk} = 0$, are posted to the sub-problem, and local probing continues having these constraints present in all the search nodes within this second branch. **3.** If forbidding the link (i, j) from the demand k does not lead to a feasible solution, the third choice is to require k and force it to use the link (i, j), i.e. $x_{ijk} = 1$, at the child nodes in the third branch. If this decision leads to failure as well, the search backtracks to higher choice points in the tree. These three choices:

1. drop demand k,
2. require demand k but forbid it from using link (i, j), and
3. require demand k and force it to use link (i, j),

fully partition the search space of the sub-problem at the master BT+CS tree node.[6] This decomposition allows us to build a local probing algorithm that

[6] The PBT decomposition above for network routing was initially suggested by Liatsos et al, and it is used in a MIP-based PBT application. A version (that does not include the first branch) of their algorithm is published in [12], which tackles a related problem to NR (having *all* demands required and a different objective function).

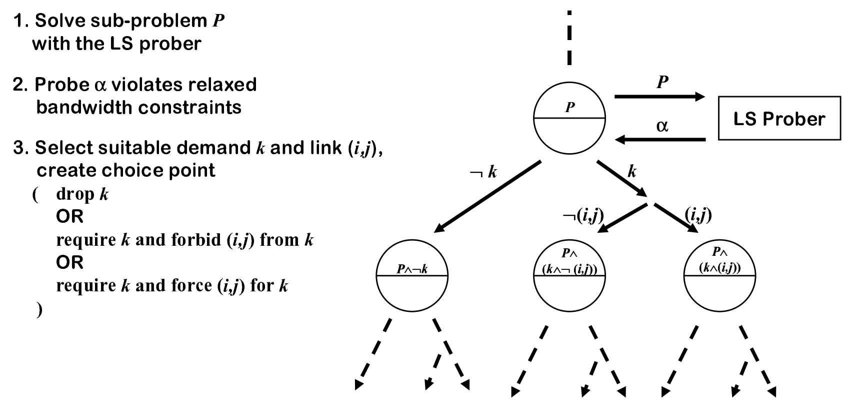
1. Solve sub-problem *P* with the LS prober
2. Probe α violates relaxed bandwidth constraints
3. Select suitable demand *k* and link (*i,j*), create choice point
(drop *k*
OR
require *k* and forbid (*i,j*) from *k*
OR
require *k* and force (*i,j*) for *k*
)
P
α
LS Prober
P
¬ k
k
¬(i,j)
(i,j)
P∧¬k
P∧ (k∧¬ (i,j))
P∧ (k∧(i,j))

Fig. 1. Illustration of local probing for NR.

satisfies the conditions for sat-completeness and also for opt-completeness. When a solution is found, a B&B cost bound CB, enforcing $\sum_{k \in K}(1 - y_k)q_k < CB$, is imposed, and local probing is continued from the root of the search tree. Due to the 'easy' constraint decisions $y_k = 0$ and $y_k = 1$ (drop/require demand) and the form of the objective function, we can immediately prune branches that are infeasible w.r.t. the cost bound. Because the algorithm can deal with the cost bound constraint as a 'hard' constraint, opt-completeness is achieved.

Several heuristics can be used in selecting the link (i, j) to be repaired among the violated links at a BT+CS search node, as well as in selecting the demand k to be dropped/required. In this implementation, the procedure selects (1) the link (i, j) that is most violated, i.e. its bandwidth availability is exceeded most; and (2) the demand k (a) routed via (i, j) and (b) having the largest bandwidth requirement from the set of demands passing over (i, j) that are *not forced to use the link* (i, j) (i.e. the set of demands that are not treated by an imposed constraint $x_{ijk} = 1$ at the master BT+CS tree nodes). These heuristics were chosen because they tend to lead the search quickly towards a feasible solution or a failure, thus reducing the size of the master BT+CS tree. Also various consistency checks are made before the local probing search is allowed to continue in the branches (reasoning with the capacity constraints and the imposed master level decisions for dropping/requiring demands and forbidding/forcing links).

2.3 Local Search Prober

The LS prober returns a probe, which is a set of paths to all non-dropped demands.[7] As an LS strategy, simulated annealing (SA, [11]) is used because it is easy to implement and can avoid local minima. At each search step, the neighbourhood operator suggests for the sub-problem a candidate neighbour that satisfies all the 'easy' constraints (recall Section 2.1).

2.4 Neighbourhood Operator of LS Prober

The neighbourhood operator procedure applies a "shuffle" approach.[8] First, a subset of demands to be re-routed are selected, and heuristically ordered for re-

[7] Note that demands can be be dropped only by the master BT+CS search decisions.
[8] Another "shuffle"-based LS algorithm for network routing is presented in [13].

routing. Then, before evaluation, the selected demands are routed separately by utilizing Dijkstra's shortest path first algorithm. The neighbourhood operator gives us the neighbour candidate in *two sets of paths*: (1) the first set S_1 is a set of paths that *together satisfy all the constraints* - including capacity constraints, and (2) the second set S_2 is a set of paths that satisfy all the 'easy' constraints but *were left outside of* S_1 by the LS heuristics because they caused capacity constraints to be violated. The procedure includes the following four steps:[9]

1. Select demands to be re-routed. From the current assignment, the paths of (1) all the demands that had paths in S_2, *and* (2) a *percentage fraction* P of the demands that had paths in S_1 are chosen for re-routing. In the latter case, paths are selected in three stages, depending on how many demands the percentage fraction allows us to select: for a randomly selected bandwidth-violated link, randomly select paths from S_1 from the following sets until P paths have been selected (start with the set D_1, and move to the next set when it is empty): **1.** Set D_1: Paths traversing the selected link. **2.** Set D_2: Paths traversing any of the links of the paths in D_1. This tends to free bandwidth in the vicinity of the problematic link. **3.** Set D_3: All the other paths in S_1. The *demands to be re-routed* are the demands of the selected paths.

2. Order demands to be re-routed. The selected demands to be re-routed are ordered for routing in three groups (the order within each group is randomized): (1) *required* demands; (2) *non-required* demands that were in S_2 in the current assignment; and (3) *non-required* demands that were in S_1 in the current assignment. This robust heuristic is used, since we prefer finding feasible solutions (routing required demands first) to optimization (minimizing unplaced total bandwidth).

3. Re-route demands. Routing is carried out in two phases. Although capacity constraints are relaxed in the LS sub-problem, the local probing performance is dependent on the LS prober producing good quality probes. Therefore, the first routing phase tries to route demands such that the *links that do not have enough bandwidth available are not seen* by the single-demand routing procedure used. If a path is found, the bandwidth availabilities of the links are updated immediately, and *the path is placed in* S_1. If a path is not found, the routing is postponed to the second phase.

In the second phase, all the selected demands, still without a path, are routed again in the same order as in the first routing phase, but now *without taking the capacity constraints into account*, i.e. also bandwidth-infeasible links are seen by the single-demand routing procedure used.[10] This phase is guaranteed, for each demand, to find a feasible path w.r.t. all constraints except the capacity constraints. *The set of paths generated in the second phase is* S_2.

4. Evaluate. The value of the neighbour candidate is the sum of (1) the unplaced bandwidth, which is the sum of the bandwidth requirements of the demands that *have paths in* S_2, and (2) the penalty component, which is the *number*

[9] The LS *initialization* includes slightly modified Steps 1 and 2, and Step 3 as it is.
[10] This causes capacity constraint violations.

of required demands in S_2 multiplied by a large constant. According to this value and the SA strategy, the assignment may or may not be accepted as the new LS assignment. (When the termination condition is met, the LS prober returns the best set of paths found throughout this prober search (the best $S_1 \bigcup S_2$).)

Single-demand routing component of neighbourhood operator. At Step 3 of the neighbourhood operator, a single demand is routed by using Dijkstra's *shortest path first* algorithm (SPF) that finds the shortest path between two nodes in a graph. The "length" of a path is the sum of the metric costs of the links in the path. The shortest path is the path where the sum of the edge costs is the lowest. Here, SPF is run over a network that *excludes* (1) the links that are forbidden for the demand to be routed, and (in the case the path must made bandwidth-feasible) (2) the links that do not have sufficient capacity. The default metric used for the cost of a link $(i, j) \in E$ is the *propagation delay* d_{ij} of the link. Since this may leave some areas of the sub-problem search space unexplored during the LS prober,[11] the propagation delays of the links are occasionally replaced by *random metrics* in the first routing phase of Step 3 of the neighbourhood operator (a resulting bandwidth-feasible path is immediately re-calculated with the default metric if it is infeasible with the delay constraints).

Routing demands with forced links. On its own, SPF cannot guarantee to satisfy forced link constraints. If SPF is run to get a path, and this path does not contain all the forced links, then an additional procedure is applied as follows. First, the forced links that are already in the path are identified. Then, the ordering they are in the path is used to define a "partial forced-link sequence" Seq. The remaining forced links are incrementally inserted in randomly selected positions in Seq such that the insertion is backtracked on failure (all possible positions in Seq are explored in the worst case). After each insertion of a forced link, SFP is applied to connect the disconnected path segments. If the completed path violates the delay constraint, backtracking to the previous insertion node occurs. The first delay-feasible path containing all the forced links is returned. If all extensions to Seq lead to failure, the same extension process is repeated for other possible permutations of Seq (in random order). The complexity of this procedure is exponential, even though the resulting path is not guaranteed to be the shortest possible. However, the average complexity is low due to the high probability of obtaining a valid path before the space is exhaustively searched.

3 Local Probing Configuration Investigation

The previous section showed a way of constructing a sat- and opt-complete local probing algorithm for the NR problem. In this section, we seek answers to the following questions: **Q1:** Can local probing be easily configured to trade-off satisfaction performance against optimization performance? **Q2:** What is the best computational balance between the BT+CS and LS components in terms of satisfaction and optimization behaviour? For problem instances with similar level of constrainedness, it is likely that certain local probing configurations are

[11] Although they would eventually be explored due to the master BT+CS decisions.

more effective than others. **Q3:** Could a single configuration remain competitive over problem instances that vary greatly in constrainedness, or are adaptive configuration mechanisms necessary?

3.1 Algorithm Parameters

Controlling the computational balance between the BT+CS and LS components: Over all the generated instances, we compared 12 local probing algorithms with different prober termination conditions that vary the balance of search effort between BT+CS and LS. The maximum number of neighbour candidate evaluations in the LS prober was set to {1, 2, 3, 6, 12, 25, 50, 100, 200, 400, 800, 1600}. We denote the local probing algorithms with these termination conditions by Probe(SA1), Probe(SA2), Probe(SA3), ... , Probe(SA1600), respectively. This range of termination conditions covers a whole range of behaviours from low (small amount of effort spent optimizing the probe by LS) to high (large amount of effort spent optimizing the probe by LS). Within the timeout selected, these configurations lead to thousands LS steps (see the averages in Tables 1 and 2). *Note that there is a key difference between Probe(SA1) and all the other configurations.* In Probe(SA1) the prober restores probe consistency w.r.t. the 'easy' constraints, but does nothing else. However, all other variants additionally perform a "shuffle" on *heuristically selected demand subsets.* At high temperatures, the neighbours are always accepted. As temperature decreases SA is more selective. **Other parameters:** In the experiments, we used a timeout of 1000 seconds. The initial temperature for the SA prober is 1000000000, and after each neighbour evaluation, the temperature is reduced by multiplying it by the cooling factor 0.9.[12] The multiplier for the penalty term for penalizing infeasible paths for required demands is the sum of the bandwidth requirements of all the demands in the instance. The probability of using randomized metrics in the first routing phase of the LS neighbourhood operator is 0.0001, and the percentage affecting the neighbourhood size is 0.1%. The algorithms were implemented on ECL^{i}PS^e 5.5, and the test were run on Pentium IV 2GHz PCs.

3.2 Problem Instances

The experiments are carried out on two different real-world network topologies, named **Network 1** and **Network 2**, with artificially generated demand data that is based on realistic demand profiles. The Network 1 topology contains 38 nodes and 86 bi-directional links, whereas the Network 2 topology is larger: 208 nodes and 338 bi-directional links. The demands are generated randomly by respecting a realistic *bandwidth profile*; features of the network topology; and the network load factor. For each network, we apply three network load factors (affecting the average bandwidth requirement), and generate 20 different sets of demands for each factor, creating 60 demand sets. For the Network 1 instances, a demand exists between each pair of nodes, ending up with 1406 demands. The

[12] A neighbour candidate is accepted if it is better than the current assignment or $p < exp^{(C_{curr} - C_{new})/T}$ where: p is a random number between 0 and 1; C_{new} is the cost of the neighbour candidate; C_{curr} is the cost of the current assignment; and T is the temperature.

Table 1. Overall satisfaction results, **Network 1**.

Configuration	Av. no. of LS steps	Solution found (#)	Proved infeasible (#)	Timeout w/o solution (#)	Rank
Probe(SA1)	965.70	499	110	51	7
Probe(SA2)	1845.48	**501**	113	46	2
Probe(SA3)	2623.46	500	**116**	**44**	**1**
Probe(SA6)	4403.40	**501**	113	46	2
Probe(SA12)	7013.62	500	114	46	2
Probe(SA25)	10219.10	500	112	48	5
Probe(SA50)	13621.08	500	108	52	8
Probe(SA100)	16569.83	**501**	109	50	6
Probe(SA200)	18950.45	500	103	57	9
Probe(SA400)	21057.47	497	93	70	10
Probe(SA800)	22456.14	491	82	87	11
Probe(SA1600)	24451.50	480	69	111	12

Table 2. Overall satisfaction results, **Network 2**.

Configuration	Av. no. of LS steps	Solution found (#)	Proved infeasible (#)	Timeout w/o solution (#)	Rank
Probe(SA1)	2176.39	479	31	150	6
Probe(SA2)	2604.19	**499**	37	124	2
Probe(SA3)	2819.54	495	**42**	**123**	**1**
Probe(SA6)	3183.90	494	41	125	3
Probe(SA12)	3691.76	495	36	129	4
Probe(SA25)	4378.13	489	32	139	5
Probe(SA50)	5074.26	479	27	154	7
Probe(SA100)	5525.13	457	23	180	8
Probe(SA200)	5878.84	443	22	195	9
Probe(SA400)	6512.35	424	11	225	10
Probe(SA800)	6883.67	400	7	253	11
Probe(SA1600)	6997.31	385	2	273	12

used network load factors were {0.6, 1.0, 1.4}. For the Network 2 instances, the demands are generated for fewer pairs of nodes, resulting in 182 demands. The network load factors used were {0.3, 0.4, 0.5}. The bandwidth profile used for generating relative bandwidth requirements of demands was different for the networks. For Network 2, it resulted in more combinatorial problems: most demands have a significant impact on the result, unlike in Network 1 whose results are dominated by a small set of demands with very large bandwidth requirements. Finally, for each demand instance we vary the proportion of required demands. The number of required demands is the number of all the demands multiplied by a *constrainedness* factor in the set {0, 0.1, 0.2, ... 1}. With these 11 factors for each of the 60 instances, we end up with 660 instances for each network, and a total of 1320 problem instances.

3.3 Constraint Satisfaction Results

Overall constraint satisfaction performance. First, we look at the overall constraint satisfaction characteristics over all instances. We are interested in **1.** how often a configuration found a solution; **2.** how often a configuration proved infeasibility; and **3.** how often an instance *remained unsolved*, i.e. how often the algorithm was terminated because of a *timeout before any solutions were found* (i.e. neither a solution nor a proof of infeasibility). The third measure is the most revealing, since it represents what is left after the first two. We therefore rank the algorithm configurations in terms of the number of timeouts. These are shown over the Network 1 and Network 2 instances in Tables 1 and 2.

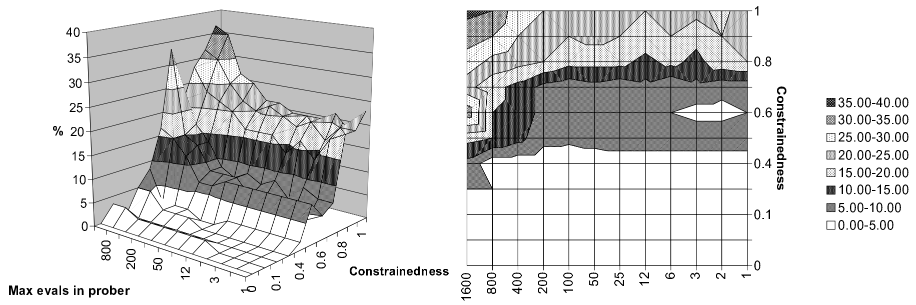

Fig. 2. Timeouts without a solution (percentage of unsolved instances), **Network 1**.

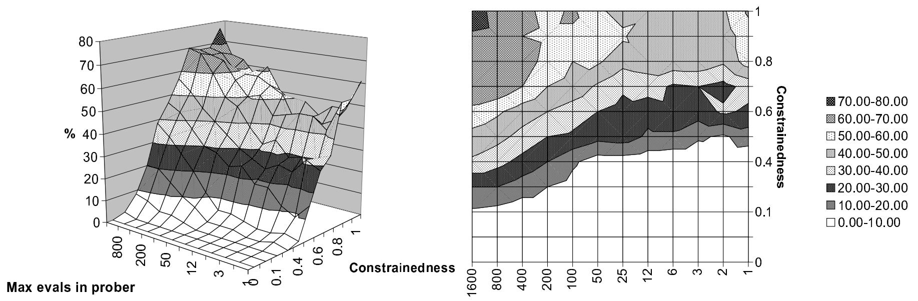

Fig. 3. Timeouts without a solution (percentage of unsolved instances), **Network 2**.

1. Solutions found. In terms of solutions found for Network 1 (Table 1), there is only a small variation between the best and worst configurations (21 instances). For Network 2 (Table 2), the variation is greater (114 instances). **2. Infeasibility proofs.** For the Network 1 instances, among the local probing algorithms, the best configuration for proving infeasibility is Probe(SA3) (116 instances). The Network 2 results indicate the same: Probe(SA3) is the best with 42 infeasibility proofs. For the Network 2 instances, Probe(SA1600) is able to prove infeasibility only in 2 cases. **3. Timeouts without a solution.** As explained above, this is the best measure for ranking constraint satisfaction performance, since it is an accurate measure of algorithm *constraint satisfaction* failure. In Network 1 cases, the best algorithm is Probe(SA3) leaving 44 instances unsolved. Probe(SA1) is seventh best, the worst configuration is Probe(SA1600). Probe(SA3) is the best configuration for constraint satisfaction also in Network 2 cases, with 123 instances unsolved. In general, the main reason for Probe(SA3)'s success in both networks, is its superiority on proving infeasibility rather than on finding solutions when there are fewer differences between configurations.

Robustness of constraint satisfaction performance as constrainedness varies. First, we want to know the the percentage of unsolved instances, when the **proportion of required demands** vary. These are illustrated in Figs. 2 and 3.[13] The results indicate that, from constraint satisfaction perspective, hybridization is useful since performance degrades at the extremes Probe(SA1) and Probe(SA1600), but that only a limited investment in LS is sufficient to obtain

[13] The rightmost graphs represent top-down views of the surfaces in the left graphs.

Table 3. Timeouts without solution over network load (number of unsolved instances).

Configuration	Network 1 load 0.6	Network 1 load 1.0	Network 1 load 1.4	Network 2 load 0.3	Network 2 load 0.4	Network 2 load 0.5
Probe(SA1)	**0**	12	39	7	49	94
Probe(SA2)	**0**	12	**34**	2	32	90
Probe(SA3)	**0**	10	**34**	1	36	**86**
Probe(SA6)	**0**	10	36	1	**30**	94
Probe(SA12)	**0**	10	36	**0**	32	97
Probe(SA25)	**0**	10	38	1	33	105
Probe(SA50)	**0**	**8**	44	**0**	48	106
Probe(SA100)	**0**	9	41	1	57	122
Probe(SA200)	1	11	45	2	64	129
Probe(SA400)	4	12	54	5	80	140
Probe(SA800)	8	17	62	9	100	144
Probe(SA1600)	9	24	78	20	107	146

the best performance from a constraint satisfaction perspective (the same is not true for optimization). The figures indicate that the best performing configuration, Probe(SA3), remains effective as problem constrainedness varies.

The **network load factor** is another way of varying problem constrainedness. Table 3 shows the numbers of unsolved instances (out of 220) for each network load factor. In the Network 1 instances, all local probing configurations up to Probe(SA100) can find a solution or prove infeasibility for all instances that have a network load of 0.6. The number of unsolved instances increases with instances of larger network loads. The bigger the network load, the more variation there is between the configurations, and fewer LS steps per prober are required to obtain the best constraint satisfaction performance. This is true also for Network 2 instances. However, the best constraint satisfaction configuration, Probe(SA3), appears to be reasonably robust also in terms of network load.

3.4 Optimization Results

The solution quality graphs in Figs. 4 and 5 show the average unplaced bandwidth[14] for each constrainedness (proportion of required demands) subset over *the instances where all algorithm configurations found a solution*, to enable a fair comparison for optimization performance.[15] In total, there were 477 such instances for Network 1, and 372 for Network 2, each out of 660.

As expected, Probe(SA1) performs worst on optimization due to the loss of LS's optimization characteristic. In the Network 1 instances, the more LS neighbour candidate evaluations performed during a prober, the better the results. However, in the Network 2 instances, the results start to get worse again when the prober evaluation limit increases beyond 12. This is due to the fact that, in terms of the demand bandwidth distribution profile, the Network 2 instances are more combinatorial (more demands significantly impact solution quality), and therefore the differences between local minima are greater. The Network 2 instances indicate that *local probing's ability to force local search away from local*

[14] The quality is scaled to the interval [0,1] such that 0 represents the best quality (all configurations routed all the demands), and 1 represents the worst quality.

[15] As constrainedness increases, more demands are required, and the costs of solutions fall because less bandwidth can be left unplaced.

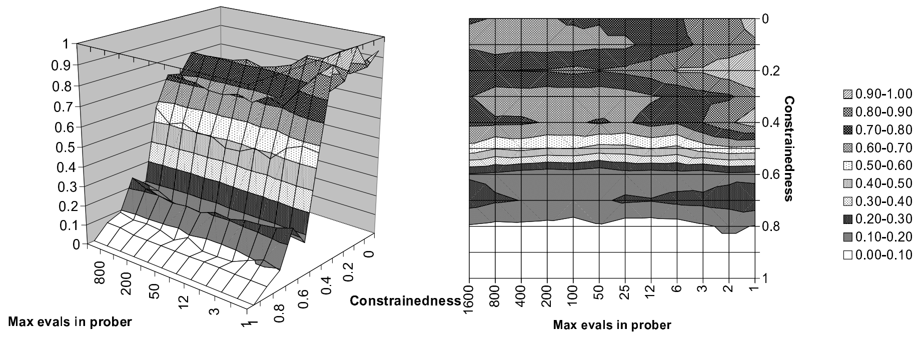

Fig. 4. Optimization, restricted subset of **Network 1**.

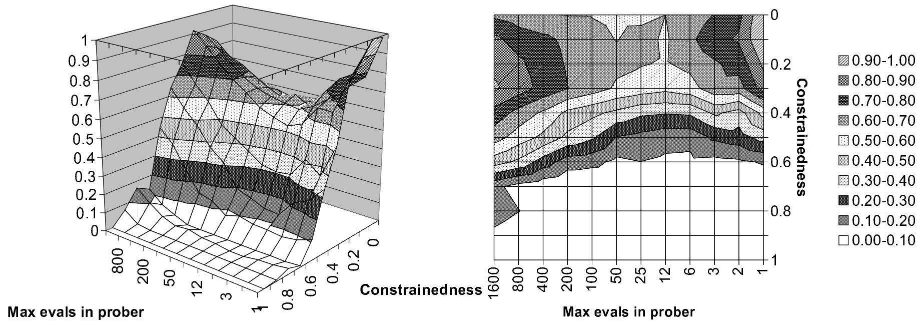

Fig. 5. Optimization, restricted subset of **Network 2**.

minima leads to significant improvements in the optimization characteristics. As the interaction of LS with BT+CS decreases, by increasing the number of the LS prober steps beyond 12, optimization performance degrades.

4 Conclusion

The main objective of this paper was to address questions on the applicability and configuration of local probing, by means of a detailed investigation of local probing performance on a new application, the NR. In the process of doing so, we established a new and versatile family of algorithms based on local probing for solving this commercially important problem.

This paper presented a *sat*- and *opt-complete* local probing hybrid for solving the NR. The local probing configurations were evaluated in a detailed experimental investigation. The hybridization balance between BT+CS and LS components was investigated by controlling the configuration of the LS prober termination condition. This successfully traded-off satisfaction performance against optimization performance **(Q1)**.[16] Relatively low LS settings (Probe(SA3) for satisfaction & Probe(SA12) for optimization) performed best overall — several thousand LS steps were sufficient per run **(Q2)**. In terms of either satisfaction only or optimization only, each configuration's relative performance remained robust over different constrainedness levels **(Q3)**. However, introducing adaptive behaviour for local probing may prove useful, because the (more BT+CS

[16] Q1, Q2 and Q3 were described in the beginning of Section 3.

oriented) configuration that was best for pure satisfaction performance was different from the (more LS oriented) configuration that was best for pure optimization performance. Future research will investigate alternative local probing neighbourhood operators and compare performance with alternative strategies for the NR problem.

Acknowledgements. We would like to thank Vassilis Liatsos, Neil Yorke-Smith and Wided Ouaja for their helpful input to this work.

References

1. F. Ajili and H. El Sakkout. A probe based algorithm for piecewise linear optimization in scheduling. *Annals of Operations Research*, 118:35–48, 2003.
2. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research*, 118:49–71, 2003.
3. T. Benoist and E. Bourreau. Improving global constraints support by local search. In *Proc. of CoSolv'03*, 2003.
4. Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical Report LIENS-95-25, École Normale Supérieure, Paris, France, 1995.
5. Y. Caseau and F. Laburthe. Heuristics for large constrained vehicle routing problems. *Journal of Heuristics*, 5(3):281–303, 1999.
6. H. El Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.
7. F. Focacci and P. Shaw. Pruning sub-optimal search branches using local search. In *Proc. of CP-AI-OR'02*, 2002.
8. J.N. Hooker, H.-J. Kim, and G. Ottosson. A declarative modeling framework that integrates solution methods. *Annals of Operations Research*, 104:141–161, 2001.
9. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139:21–45, 2002.
10. O. Kamarainen and H. El Sakkout. Local probing applied to scheduling. In *Proc. of CP'02*, pages 155–171, 2002.
11. S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
12. V. Liatsos, S. Novello, and H. El Sakkout. A probe backtrack search algorithm for network routing. In *Proc. of CoSolv'03*, 2003.
13. S. Loudni, P. David, and P. Boizumault. On-line resources allocation for ATM networks with rerouting. In *Proc. of CP-AI-OR'03*, 2003.
14. B. Mazure, L. Saïs, and É. Grégoire. Boosting complete techniques thanks to local search. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.
15. S. Prestwich. A hybrid search architecture applied to hard random 3-sat and low-autocorrelation binary sequences. In *Proc. of CP'00*, pages 337–352, 2000.
16. L. Rousseau, M. Gendreau, and G. Pesant. Using constraint-based operators to solve the vehicle routing problem with time windows. *J. Heuristics*, 8:45–58, 2002.
17. M. Sellmann and W. Harvey. Heuristic constraint propagation: Using local search for incomplete pruning and domain filtering of redundant constraints for the social golfer problem. In *Proc. of CP-AI-OR'02*, 2002.
18. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. of CP'98*, pages 417–431, 1998.
19. M. Wallace and J. Schimpf. Finding the right hybrid algorithm - a combinatorial meta-problem. *Annals of Mathematics and Artificial Intelligence*, 34:259–269, 2002.