Discrete Applied Mathematics 40 (1992) 45–72
North-Holland

Steiner's problem in graphs: heuristic methods

Stefan Voß

FB 1/FG Operations Research, Technische Hochschule Darmstadt, Hochschulstraße 1, D-6100 Darmstadt, Germany

Received 27 August 1989
Revised 13 May and 12 September 1990

Abstract

Voß, S., Steiner's problem in graphs: heuristic methods, Discrete Applied Mathematics 40 (1992) 45–72.

Real world problems arising in the layout of connection structures in networks as e.g. in VLSI design may often be decomposed into a number of well-known combinatorial optimization problems.

Steiner's problem in graphs is included within this context. According to its complexity one is interested in developing efficient heuristic algorithms to find good approximate solutions. Here a comparison of various heuristic methods for Steiner's problem in graphs is presented.

1. Introduction

Given a graph with weights on its edges *Steiner's problem in graphs* (*SP*) is to determine a minimum cost subgraph spanning a set of specified vertices.

More formally, let $G=(V,E)$ be an undirected connected graph with vertex set V, edge set E, and nonnegative weights associated with the edges. Given a set $Q \subseteq V$ of *basic vertices* SP is to find a minimum cost subgraph of G such that there exists a path in the subgraph between every pair of basic vertices. In order to achieve this minimum subgraph additional vertices from $V-Q$, so-called *Steiner vertices*, may be included.

Real world problems arising in the layout of connection structures in networks as e.g. in VLSI design may often be decomposed into a number of well-known combinatorial optimization problems. SP is included within this context (cf. [14]).

Two well-known special cases of SP are polynomially solvable. If $|Q|=2$ the

Correspondence to: Dr. S. Voß, FB 1/FG Operations Research, Technische Hochschule Darmstadt, Hochschulstraße 1, D-6100 Darmstadt, Germany.

problem reduces to a shortest path problem and in the case where $Q = V$ SP reduces to the minimum spanning tree problem. However, in the general case SP is an NP-hard problem. For a comprehensive survey on SP the reader is referred to the excellent paper of Winter [38] as well as Voß [33].

According to SP's complexity one is interested in developing efficient heuristic algorithms to find good approximate solutions. Following Winter [38] one of the most important open problems with SP worthwile to explore is a comparison of various heuristic methods for SP. So this paper is intended to present an overview on approximate solution methods for SP (Section 2). (For a comparison of specialized heuristics when distances are restricted to be rectilinear see [28].)

Additional topics included within this survey on heuristics for SP are:

- comparability of the algorithms (Section 3),
- worst case analysis results (Section 3),
- improvement procedures (Section 4),
- computational results (Section 5).

Some ideas for further research will be given in the last section.

2. Heuristics

In the following several heuristics for SP will be compiled. For ease some notation is used:

- $G = (V, E)$: given graph with vertex set $V = Q \cup S$ and edge set E where Q is the set of basic vertices and S is the set of possible Steiner vertices (with $Q \cap S = \emptyset$),
- c_{ij}: nonnegative cost or weight of edge $(i, j) \in E$,
- d_{ij}: weight of a shortest path between i and j in G,
- $P(i, j)$: a shortest path between i and j in G,
- $d(i, T) = \min\{d_{ij} \mid j \in V_T\}$: weight of a shortest path between a vertex i and the subgraph $T = (V_T, E_T)$ of G ($d(i, T \cap Q) = \min\{d_{ij} \mid j \in V_T \cap Q\}$).

For any subgraph $T = (V_T, E_T)$ of G we define Z_T as the sum of its edges' weights. For clarity a connected subgraph T that includes a path between every pair of basic vertices will be denoted as a feasible solution. Correspondingly an optimal solution is a feasible solution of minimum cost Z_{opt}.

Remark. Wlog we may assume $3 \leq |Q| \leq |V| - 1$.

Whenever one of the heuristics starts with a single basic vertex w this vertex may be viewed as a starting vertex or root of the solution. If not stated otherwise w will be chosen arbitrarily from Q.

In addition whenever ties occur they are broken arbitrarily if not stated otherwise.

Before describing the heuristics in detail some of their basic ingredients will be discussed.

A central theme to all heuristics is the use of some principles known from

algorithms to find minimum spanning trees together with the computation of shortest paths. Two main ideas for constructing feasible solutions for SP may be distinguished:

- *INSERTION*: Start with a "partial solution" $T=(\{w\},\emptyset)$ consisting of a single basic vertex. T will be expanded to a feasible solution by successively inserting all basic vertices e.g. through the computation of at most $|Q|$ shortest paths.

This idea corresponds to the proceeding of Prim's minimum spanning tree algorithm [24].

- *COMPONENT CONNECTING*: Start with a "partial solution" $T=(Q,\emptyset)$ consisting of $|Q|$ singleton components. T will be expanded to a feasible solution by the computation of shortest paths successively linking together all components.

When in each step two components will be connected by a shortest path ("2BASIC") this idea corresponds to the proceeding of Kruskal's minimum spanning tree algorithm [18]. Another variant tries to connect three components in each iteration ("3BASIC").

Modifications or combinations of these two approaches are possible. A more evolved classification will be given after the description of algorithms for SP known from the literature.

The first two heuristics to be considered are quite simple ones (cf. [31]).

SPATH: *Shortest paths.*

(1) Start with $T=(\{w\},\emptyset)$ consisting of a single basic vertex.

(2) For all basic vertices $i\in Q-\{w\}$ add the vertices and edges of $P(w,i)$ to T.

MST+P: *Minimum spanning tree and pruning.*

(1) Construct a minimum spanning tree $T=(V_T,E_T)$ of G.

(2) **While** there exists a leaf of T being a Steiner vertex **do** delete that leaf and its incident edge.

Although being quite simple both heuristics may outperform more evolved methods (compare Section 3). Based on MST+P a modification has also been proposed which consists of iteratively repeating steps (1) and (2) with the edge weights modified according to some penalty function (cf. [10]; for an additional modification see [33]).

Independently the following heuristic has been developed by several authors [6, 17, 23].

SDISTG: *Shortest distance graph.*

(1) Construct the complete graph with vertex set Q and each edge having the weight of a shortest path between the corresponding vertices in G.

(2) Construct a minimum spanning tree of this (shortest distance) graph.

(3) Replace each edge of the tree by its corresponding shortest path in G.
(4) Delete edges and Steiner vertices such that no cycles exist and all leaves are basic vertices.

If $T=(V_T, E_T)$ is the resulting graph of step (3) then step (4) of SDISTG may be further specified as:

(4.1) Construct a minimum spanning tree $T'=(V_T', E_T')$ of T.
(4.2) **While** there exists a leaf of T' being a Steiner vertex **do** delete that leaf and its incident edge.

A FORTRAN code of SDISTG is listed in [20]. SDISTG has received much attention in the literature with respect to an efficient implementation using sophisticated data structures (cf. [15, 16, 21, 36, 37, 41]). When constructing the complete graph for vertex set $Q \cup S'$ for all subsets $S' \subseteq S$ with $0 \leq |S'| \leq k$ in step (1) we get another heuristic method (cf. [23,30]). For $k=|Q|-2$ this results in an enumerative approach to solve SP exactly (with running time at least exponential). Especially for $|Q|=3$ and $k=1$ we get the algorithm 3BASIC which is polynomial in the input. This algorithm may also be applied for the case of uniting optimally three components each containing at least one basic vertex (compare the idea of COMPONENT CONNECTING, a somewhat different description of 3BASIC is given in [5]).

In the following some insertion methods will be presented. Again we will start with a quite simple one.

ARINS: *Arbitrary insertion.*
(1) Start with $T=(\{w\}, \emptyset)$ consisting of a single basic vertex.
(2) **Repeat** choose an arbitrary basic vertex p^* not in V_T, and find a nearest vertex $v^* \in V_T$, i.e., a vertex with $d_{v^*p^*}=\min\{d_{vp^*} \mid v \in V_T\}$ and add the vertices and edges of $P(v^*, p^*)$ to T
until T contains all basic vertices.

Takahashi and Matsuyama [31] were the first to describe a cheapest insertion method which is a more evolved variant of ARINS.

CHINS: *Cheapest insertion.*
(1) Start with $T=(\{w\}, \emptyset)$ consisting of a single basic vertex.
(2) **Repeat** find nearest vertices v^* and p^* with $v^* \in V_T$ and p^* being a basic vertex not in V_T, i.e., vertices with $d_{v^*p^*}=\min\{d_{vp} \mid v \in V_T, p \in Q - V_T\}$ and add the vertices and edges of $P(v^*, p^*)$ to T
until T contains all basic vertices.

CHINS is also treated in [12, 29, 34]. One of the most interesting aspects with

CHINS is the influence of how the root w is chosen. One idea to show this might result in a modified algorithm.

CHINS-A: *Cheapest insertion (all roots).*

(1) Apply CHINS for every basic vertex defined as the root w.

(2) Redefine T as the cheapest of all solutions evaluated in (1).

Another approach consists of choosing a suitable subtree of G instead of a single vertex as a starting component (cf. [5]). In addition some local optimization may be applied in each iteration.

CHINS-3: *Cheapest insertion (with 3BASIC).*

(1) Apply 3BASIC to all subsets of Q with three elements.

(2) Initialize T as the cheapest of all solutions evaluated in (1).

(3) **Repeat**

(3.1) Find nearest vertices v^* and p^* with $v^* \in V_T$ and p^* being a basic vertex not in V_T, i.e., vertices with $d_{v^*p^*} = \min\{d_{vp} \mid v \in V_T, p \in Q - V_T\}$.

(3.2) **For all** paths in T connecting v^* with a vertex i s.t. all vertices of this path but v^*, i are Steiner vertices of degree 2 (with respect to T) **do** remove this path without v^*, i from T to disconnect T into two components T_{v^*} and T_i (containing v^* and i, respectively) and apply 3BASIC to these two components together with p^* viewed as a third component.

(3.3) Redefine T as the cheapest of all solutions evaluated in (3.2), i.e., the corresponding path is eliminated and vertices and edges are added according to 3BASIC making T a connected subgraph containing p^*

until T contains all basic vertices.

One idea for improving the insertion methods described above with respect to solution quality could consist of a similar approach as described for SDISTG (with obviously increased computation times): Replace Q by a modified set of basic vertices $Q \cup S'$ for all subsets $S' \subseteq S$ with $0 \leq |S'| \leq k$ (with k being a positive integer).

In addition CHINS may be related to SDISTG as follows. If in step (2) vertices are chosen according to $d_{v^*p^*} = \min\{d_{vp} \mid v \in V_T \cap Q, p \in Q - V_T\}$ then CHINS corresponds to steps (1)–(3) of SDISTG, since it turns out to be an implementation of Prim's algorithm for the computation of a minimum spanning tree of the complete graph with vertex set Q as given in step (1) of SDISTG. In addition even step (4) of SDISTG may be applied to the solution found with CHINS (cf. Section 4).

Before presenting the next heuristic a motivation on its principle seems to be necessary.

If the set V_T of an optimal solution is known for given data of SP the remaining problem consists of polynomially computing a minimum spanning tree of the

subgraph of G induced by V_T. Since V_T contains Q and some Steiner vertices one approximate approach for determining these Steiner vertices uses a heuristic measure or heuristic function (cf. [9,25]). Such a function will be defined by assigning a real value $f(v)$ to each vertex $v \in V$ depending on the given data of SP. A general description of a corresponding heuristic is as follows.

HEUM: *Heuristic measure.*

(1) Start with $T = (Q, \emptyset)$ comprising $|Q|$ basic vertices (subtrees of the final subgraph).

(2) **While** T is not connected **do**
choose a vertex v using a heuristic function f and unite the two components of T which are nearest to v by combining them with v via shortest paths (the vertices and edges of these paths are added to T).

Generally, the heuristic function f will be minimized (where all vertices may come into consideration independently of either being an element of V_T or not). By choosing a suitable heuristic function f, e.g. SDISTG may be viewed as a special form of HEUM:

$$f(v) := \min_{\substack{0 \le i,j \le \sigma \\ i \ne j}} \{d(v, T_i \cap Q) + d(v, T_j \cap Q) \mid T_0, \ldots, T_\sigma \text{ are the components of } T\}.$$

Up to now the most promising way is to choose f according to

$$f(v) := \min_{1 \le t \le \sigma} \left\{ \frac{1}{t} \cdot \sum_{i=0}^{t} d(v, T_i) \mid T_0, \ldots, T_\sigma \text{ are the components of } T \right\}$$

(cf. [9,25,26]). Intuitively $f(v)$ gives a measure of proximity of v to elements of a subset of vertices in T, or the least average distance of v to a set of vertices in T, respectively. In [25] rules are developed for breaking ties concerning this definition of f. In what follows (especially in our implementation of HEUM below) we propose the use of some special ordering:

$$f(v) := \min_{1 \le t \le \sigma} \left\{ \frac{1}{t} \cdot \sum_{i=0}^{t} d(v, T_i) \mid T_0, \ldots, T_\sigma \text{ are the components of } T \text{ s.t. } d(v, T_i) \le d(v, T_j)\ \forall i, j \in \{0, \ldots, \sigma\}, i < j \right\}.$$

In HEUM having selected v this vertex is used to unite two components of the solution to be evaluated. However, this approach may be modified.

HEUM-3: *Heuristic measure (with 3BASIC).*

(1) Start with $T = (Q, \emptyset)$ comprising $|Q|$ basic vertices (subtrees of the final subgraph).

(2) **While** T consists of at least three disconnected components **do**
choose a vertex v using a heuristic function f and unite the three com-

ponents of T which are nearest to v by combining them with each other by applying 3BASIC (and adding the vertices and edges of that solution to T).

(3) **If** T is not connected **then**
unite the two components of T by combining them with each other by a shortest path (the vertices and edges of this path are added to T).

Note that the vertex v chosen in step (2) need not either be an element of V_T nor be added to V_T in subsequent steps or iterations. In [5] an algorithm is described which corresponds to HEUM-3 by choosing f according to

$$f(v) := \sum_{i=0}^{2} d(v, T_i)$$

where $T_0, \ldots, T_\sigma$ are the components of T s.t. $d(v, T_i) \leq d(v, T_j)\ \forall i, j \in \{0, \ldots, \sigma\}, i < j$ (this algorithm will be referred to as HEUM-3 subsequently). Given the components $T_0, \ldots, T_\sigma$ in the same way and choosing f according to

$$f(v) := \sum_{i=0}^{1} d(v, T_i)$$

which applies the idea of 2BASIC instead of 3BASIC we get a proceeding similar to that of Kruskal's minimum spanning tree algorithm.

Wong [40] describes a dual ascent algorithm based on a multicommodity network flow formulation of SP in directed graphs (SPD). SPD is to find a minimum cost directed subgraph of a given graph that contains a directed path between a root node and every basic vertex. Any instance of SP may be transformed into an instance of SPD by replacing each undirected edge $(i, j) \in E$ with two arcs $[i, j]$, $[j, i]$ directed opposite each other both having the same cost c_{ij}. An arbitrary basic vertex is stressed as the root w.

Based on the formulation of SPD as a multicommodity network flow problem and the dual of the corresponding linear programming formulation we get the following heuristic algorithm. The feasible solution T in this algorithm is assumed to be a directed subgraph of the given graph transformed into an instance of SPD. A retransformation simply ignors the orientation of each arc to result in a feasible solution of SP.

DUASC: *Dual ascent.*

(1) Start with $T = (V_T, E_T)$ comprising $|Q|$ basic vertices, i.e., $T = (Q, \emptyset)$. Let w be an arbitrary fixed element from Q.

(2) **Repeat**

(2.1) Select a strongly connected component $T_c = (V_c, E_c)$ of T with
(i) $V_c \cap (Q - \{w\}) \neq \emptyset$,
(ii) there exists no directed path in T from any basic vertex (including w) not in V_c to any vertex of V_c.

(2.2) Select an arbitrary basic vertex $q \in V_c$ and define $CS(q) := \{[i,j] \in E \mid [i,j] \notin E_T$, in T exists a directed path from j to q but not from i to $q\}$ ($CS(q)$ may be viewed as some kind of cutset).

(2.3) Find an arc $[i^*,j^*]$ from $CS(q)$ with $c_{i^*j^*} = \min\{c_{ij} \mid [i,j] \in CS(q)\}$.

(2.4) **For all** $[i,j] \in CS(q)$ **do** $c_{ij} := c_{ij} - c_{i^*j^*}$.

(2.5) Update T by setting $V_T := V_T \cup \{i^*\}$ and $E_T := E_T \cup \{[i^*,j^*]\}$

until T contains a path from w to all basic vertices.

(3) Delete arcs while preserving feasibility of T such that no cycles exist and all leaves are basic vertices.

As in SDISTG step (3) may be further specified:

(3.1) **While** there exists a Steiner vertex $v \in V_T$ s.t. no path from w to v exists in T **do**
delete v and its incident edges.

(3.2) Construct a minimum directed spanning tree $T' = (V_T', E_T')$ of T.

(3.3) **While** there exists a leaf of T' being a Steiner vertex **do**
delete that leaf and its incident edge.

DUASC finds a feasible solution to any instance of SP at which a lower bound (corresponding to the dual variables) may also be calculated giving information on the quality of the feasible solution (cf. [22,40]).

For the sake of completeness let us describe two additional approaches from the literature.

Beside SDISTG Plesnik [23] describes a second heuristic which may be viewed as a recursive algorithm using some kind of contraction. It consists of shrinking subgraphs containing at least one basic vertex to a single pseudonode which again is considered to be a basic vertex for a modified problem instance.

CONTR: *Method of contraction.*

(1) Start with $T = (Q, \emptyset)$ comprising $|Q|$ basic vertices (subtrees of the final subgraph).

(2) Find a minimum cost edge incidence to a basic vertex, i.e., an edge (i^*,j^*) with $c_{i^*j^*} = \min\{c_{ij} \mid (i,j) \in E, i \in Q \vee j \in Q\}$.

(3) Define

$$c'_{ij} := \begin{cases} \max\{c_{ij} - 2 \cdot c_{i^*j^*}, 0\}, & i,j \in Q, \\ c_{ij}, & i,j \notin Q, \\ c_{ij} - c_{i^*j^*}, & \text{otherwise}, \end{cases}$$

and $T_c = (V_c, E_c)$ with $E_c := \{(i,j) \in E \mid c'_{ij} = 0\}$ and V_c the corresponding vertex set.

(4) Solve an SP for each component $T_1, \ldots, T_\sigma$ of V_c with respect to the

basic vertices in this component (e.g. with CHINS) and add the corresponding vertices and edges to T.
If $\sigma = 1$ **then** return (or stop, respectively).

(5) Contract each component T_k into a single basic vertex t_k ($k = 1, \ldots, \sigma$), i.e., edges incident to any vertex of a component T_k are made incident to the corresponding vertex t_k (multiple edges and loops may be neglected). With $Q' := \{t_1, \ldots, t_\sigma\}$ this leads to a modified graph $G' = ((V - V_c) \cup Q', E - E_c)$ with weights c'.

(6) Solve an SP for Q' in G' (by recursion, i.e. apply CONTR). Reconnect this solution and V_c by adding no more than γ_k edges of length $c_{i^*j^*}$ for each vertex t_k to T where γ_k is the number of edges incident to t_k (the nature of the contraction in (5) makes this always possible).

Aneja [1] formulates SP as an equivalent set covering problem with a number of constraints growing exponentially with the size of given data or problem instances. However, these constraints may be handled implicitly within a row generation scheme. A more general description makes use of any heuristic for the transformed formulation.

SETCOV: *Set covering.*

(1) Transform the given data of SP into an equivalent set covering problem.

(2) Solve the set covering problem by any heuristic available for this problem.

3. Comparison of the algorithms

In the last section heuristic algorithms for SP have been described. By distinguishing the two main topics INSERTION and COMPONENT CONNECTING some kind of classification of the algorithms may be given. Beside the partial solution to start with there exist three building blocks used within this frame.

- 1BASIC: Given one component look for a cheapest edge with one vertex within and the other one outside the component and enlarge the component by at least this edge.
- 2BASIC: Given two components (a single basic vertex not yet part of a partial solution may be viewed as a component) connect them by a shortest path.
- 3BASIC: Given three components connect them by use of some shortest paths.

However, the main classification criterion is based on the knowledge which is used in each algorithm for the calculation of a feasible solution. Together with the specific implementation using the INSERTION or the COMPONENT CONNECTING idea and including one of the building blocks described above we get different

Table 1
Classification of heuristics

Special emphasis on	INSERTION methods		COMPONENT CONNECTING methods		
	Prim-related 2BASIC	3BASIC	1BASIC	Kruskal-related 2BASIC	3BASIC
no specification	MST + P			MST + P	
basic vertices	SPATH, SDISTG		SDISTG		
basic vertices and partial solution	ARINS(-I) CHINS(-I) CHINS-A(-I)	CHINS-3(-I)	DUASC(-I)		
partial solution			CONTR	HEUM(-I)	HEUM-3(-I)

algorithms. This results in Table 1 classifying existing algorithms for SP (SDISTG as well as MST + P may be implemented by either using Prim's or Kruskal's algorithm). The supplement I refers to the same algorithm combined with an improvement procedure (see Section 4).

Comparability results are twofold. Concerning computational experiences we refer to Section 5. The other aspect is on theoretical investigations. Again mainly two criteria are considered, the worst case error ratio and the time complexity. In Table 2 known results on these two points of view are shown. (Whenever in Table 2 a value for the worst case error ratio is given this bound is tight, i.e., Z_T/Z_{opt} may equal this value for a suitable feasible solution T.) Results on SETCOV and DUASC are not known from the literature.

From Table 2 we may deduce that all but the two simplest heuristics satisfy the

Table 2
Worst case results

Algorithm	$Z_T/Z_{\text{opt}} \leq$	Complexity	Remarks
SPATH	$\lvert Q\rvert - 1$	$O(\lvert V\rvert^2)$	cf. [19, 31]
MST + P	$\lvert S\rvert + 1$	$O(\lvert V\rvert^2)$	cf. [31]; given a feasible solution the worst case ratio may be expressed with the number of Steiner vertices within this solution, cf. [33]
SDISTG	$2(1 - 1/q^*)$	$O(\lvert Q\rvert \cdot \lvert V\rvert^2)$	$q^* \leq \lvert Q\rvert$ denotes the number of leaves in a feasible solution, cf. [6, 17, 23]; the complexity may be improved at least to $O(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$, cf. [21]
CHINS	$2(1 - 1/\lvert Q\rvert)$	$O(\lvert Q\rvert \cdot \lvert V\rvert^2)$	cf. [12, 31]
CHINS-A	$2(1 - 1/\lvert Q\rvert)$	$O(\lvert Q\rvert^2 \cdot \lvert V\rvert^2)$	
CHINS-3	$2(1 - 1/\lvert Q\rvert)$	$O(\lvert Q\rvert \cdot \lvert V\rvert \cdot \lvert E\rvert \log \lvert E\rvert)$	the complexity may be expressed with a term using the maximum vertex degree of G, cf. [5]
HEUM	$2(1 - 1/\lvert Q\rvert)$	$O(\lvert V\rvert^3)$	cf. [9, 35]
HEUM-3		$O(\lvert Q\rvert^2 \cdot \lvert E\rvert \log \lvert E\rvert)$	cf. [5]
CONTR	$2(1 - 1/\lvert Q\rvert)$	$O(\lvert V\rvert^3)$	cf. [23]

same general worst case error ratio. So it is important to see whether these algorithms are comparable or not in the sense that they outperform each other in quality of solution for different problem instances, i.e., given data of different graphs.

Let Z_i and Z_j denote the objective function values derived by algorithm A_i and A_j, respectively. A_i and A_j are called *incomparable* if instances with $Z_i < Z_j$ as well as instances with $Z_i > Z_j$ exist. If A_i and A_j are incomparable independently of how ties are broken when both algorithms are applied to given instances we call them *strongly incomparable*. For some of the algorithms above, namely SDISTG, CHINS, CHINS-3, and HEUM-3 we get (cf. [36]) that any two of them are strongly incomparable. However, this definition of incomparability does not give any idea on which algorithms could be preferred against others because even the simplest ones SPATH and MST + P have to be added to this list of strongly incomparable algorithms (cf. [33]).

In addition it may be shown that CONTR and SDISTG give the same results on special instances (cf. [33]).

4. Improvement procedures

Only a very small number of improvement procedures has been developed within the framework of SP. The first idea is some kind of pruning routine like in MST + P (step (2)). A modification of this routine is included within step (4) of SDISTG (cf. steps (4.1) and (4.2) below SDISTG). However, the most promising approach consists of applying MST + P to a modified graph.

I-MST + P: *Improvement procedure (with MST + P).*

(1) Let $T = (V_T, E_T)$ be a feasible solution of SP. The subgraph of G induced by V_T will be defined as G_T.

(2) Construct a minimum spanning tree $T' = (V'_T, E'_T)$ of G_T.

(3) **While** there exists a leaf of T' being a Steiner vertex **do** delete that leaf and its incident edge.

In the following I-MST + P will be the only algorithm applied together with the heuristics described in the preceding section (cf. [26]). Whenever an algorithm is reoptimized via I-MST + P this will be denoted by the supplement I (e.g. CHINS-I or HEUM-I).

Implicitly another improvement procedure may be derived from steps (4) and (5) of CHINS-3, too. This may also hold as an idea for an interchange procedure in connection with simulated annealing. Such a probabilistic exchange procedure tries to overcome the deficiency of bad local optima by allowing temporary deteriorations of actual solutions. The most promising framework for an annealing algo-

rithm on SP makes use of some interchange heuristic which adds/drops either edges or Steiner vertices to/from a feasible solution. We briefly give an idea for the transformation of a feasible solution $T=(V_T, E_T)$ to another one. (Because of the assumption of nonnegative weights we may assume that T is a tree.) Up to now this has only been realized for the more general directed version of SP (SPD, cf. [27, 33]). Recently the application of a genetic algorithm to SP has also been proposed (cf. [11]).

TRAFO: *Transformation.*

(1) Select randomly an edge $(i,j) \in E_T$ and delete it from E_T.

(2) **For** $h=i,j$ **do**
select a path in T connecting h with a vertex h^* s.t. all vertices of this path but h^* are Steiner vertices of degree 2 (with respect to T) and remove this path from T (with h^* remaining in V_T).

(3) Let T_i and T_j be the two disconnected components of T. Find nearest vertices i^* and j^* belonging to T_i and T_j, respectively, and add the vertices and edges of a shortest path between i^* and j^* to T.

I-MST + P and TRAFO assume the existence of a feasible solution derived by any heuristic algorithm. Improvements in a more general sense may also be obtained by applying some efficient preprocessing routines to reduce given input data. Such reduction techniques are described in e.g. [2, 7, 8, 32]. Note that preprocessing routines, however, are no improvement procedures by itselves.

In a more general scheme these techniques may also be applied within any of the algorithms described in Section 2. Whenever at least one edge or vertex has been added to a partial solution some reductions may be applied before going to the next iteration (cf. [33]). As mentioned above, however, in this paper we will restrict ourselves on the application of I-MST + P since our main objective lies in the heuristics itselves. Nevertheless the general scheme of combining heuristics with reduction techniques seems to be a quite promising area of research.

5. Computational results

To the best of our knowledge up to now computational comparisons on heuristics for SP have only been presented by Rayward-Smith and Clare [26] (see also [38]), and very recently in [39]. In all other contributions algorithms have at most been tested on a limited number of test problems and not compared to other heuristics. In this section the results of [26] will be summarized. Then additional computational experiments we have undertaken will be reported.

Rayward-Smith and Clare [26] compared three algorithms with each other: SDISTG, CHINS-I, and HEUM-I (with some additional remarks on CHINS and HEUM).

All test problems treated in [26] are randomly generated ones. The first set of problems corresponds to problems of [3, 4] which have been generated following a scheme outlined in [1]. Given $|V|$ and $|E|$ first a random spanning tree over V is generated. Then additional edges up to $|E|$ are added randomly and real weights uniformly distributed in the range [1, 10] are assigned to each edge. Basic vertices are determined randomly with $|Q| \leq \frac{1}{2}|V|$. Table 3 shows the results corresponding to this set of test problems (the last problem is not contained in [3, 4]). For all but two problems (marked with *) the optimal solution is known and the values of Table 3 give a "%-error" or deviation from the optimum calculated by the formula:

$$\text{deviation (in \%)} = \frac{Z_T - Z_{\text{opt}}}{Z_{\text{opt}}} \cdot 100.$$

Whenever an optimal solution value is unknown Z_{opt} is defined as the cost of the best known feasible solution. In addition the average and the maximum deviation over all problems are calculated. For each algorithm the achieved number of best results with respect to all computed solutions is presented, too.

Although the test problems of Table 3 are quite sparse graphs (with the last prob-

Table 3
Results of Rayward-Smith and Clare [26]

Data			% deviation from optimal solution		
$\|V\|$	$\|E\|$	$\|Q\|$	SDISTG	CHINS-I	HEUM-I
50	63	9	0.0	0.0	0.0
		13	8.4	0.0	0.0
		25	1.4	0.0	0.0
50	100	9	8.5	5.1	5.1
		13	4.9	0.0	0.0
		25	4.9	3.3	1.6
75	94	13	0.0	0.0	0.0
		19	0.0	0.0	0.0
		38	2.3	0.0	0.0
75	150	13	14.0	4.7	4.7
		19	2.3	2.3	2.3
		38	0.0	0.0	0.0
100	125	17	6.1	7.9	4.2
		25	1.3	2.6	0.4
		50	2.2	0.0	0.0
100	200	17*	7.9	3.1	0.0
		25	5.3	3.8	3.1
		50	4.6	1.8	0.0
100	500	25*	1.6	0.0	0.0
Average deviation			3.9	1.8	1.3
Maximum deviation			14.0	7.9	5.1
Best results (out of 19)			5	13	19
Optimal solutions (out of 17)			4	9	10

lem as an exception) it may be concluded that both CHINS-I and HEUM-I outperform SDISTG in the quality of solutions. As reported in [26] all computation times lie within a range of ten minutes (on an ICL 1904T computer, coded in Algol 68-R).

In a second sequence of experiments random graphs with $|V| \in \{10, 20, 40, 80\}$ have been generated as follows (cf. [26]). Two probabilities are fixed, $p1$ for the probability that an edge exists between any two vertices, and $p2$ for the probability that a vertex is defined as a basic vertex. If the resulting graph is connected real edge weights will be assigned either uniformly distributed (U) in the range (0,1] or by a normal distribution (N) with mean 0.5 and standard deviation 0.125. Table 4 summarizes the results of SDISTG, CHINS-I, and HEUM-I on 1500 graphs following this scheme. The number of best results (in %) and the maximum deviation (max. dev.) on each sample are listed (optimal solutions are unknown). Again SDISTG is outperformed by the two other algorithms, and HEUM-I gives better results than CHINS-I (although examples occur where CHINS-I beats HEUM-I). With these data in [26] some remarks on the impact of improvements with HEUM-I and CHINS-I versus HEUM and CHINS are also given saying that improvements occur quite often but are rather small (whereas improvements increase for graphs with $|V|$ increasing).

In what follows additional computational experiments are undertaken comparing more than the three algorithms investigated above. In order to get comparative results the following algorithms and modifications, respectively, have been implemented:

SPATH, MST+P, SDISTG,
ARINS, CHINS, CHINS-A, CHINS-3,
ARINS-I, CHINS-I, CHINS-A-I, CHINS-3-I,
HEUM, HEUM-I, HEUM-3, HEUM-3-I,
DUASC, DUASC-I.

Table 4
Results of Rayward-Smith and Clare [26]

					SDISTG		CHINS-I		HEUM-I	
$\|V\|$	$p1$	$p2$	No. of examples	Distribution	% best results	max. dev.	% best results	max. dev.	% best results	max. dev.
10	0.25 ... 1.0	0.5	200	U	86.5	20.9	98.0	9.5	100	0.0
20	0.25 ... 1.0	0.25 ... 0.75	300	U	72.3	14.2	95.7	10.2	100	0.0
40	0.125 ... 1.0	0.125 ... 0.75	200	U	43.0	24.9	84.5	24.9	98.0	4.9
80	0.125 ... 0.25	0.125 ... 0.9	50	U	18.0	20.5	78.0	5.3	90.0	1.7
10	0.25 ... 1.0	0.5	200	N	90.5	27.3	97.0	12.1	99.5	1.6
20	0.25 ... 1.0	0.25 ... 0.75	300	N	82.3	12.9	93.7	8.8	99.7	3.8
40	0.125 ... 1.0	0.125 ... 0.75	200	N	62.0	26.3	85.5	16.4	99.0	1.6
80	0.125 ... 0.25	0.125 ... 0.9	50	N	46.0	27.6	74.0	4.9	100	0.0

All algorithms have been implemented in FORTRAN 77 on an AT Personal Computer (10 Mhz). In different kinds of experiments we generated data with random edge weights, Euclidean, and rectilinear distances to run the algorithms on. The first set of test problems is based on three special planar graphs (developed for having a well-structured data base for testing the algorithms). All these graphs start with small numbers of nodes and edges (9 and 20, 6 and 9, 5 and 13, respectively). They are successively enlarged by adding vertices and edges following the structure of Figs. 1–3 (for the case where $|V| = 37$, 28, and 26, respectively). Edge weights are randomly generated integers in the range [1, 99]. Table 5 shows results for graphs according to this scheme. Basic vertices are ranging from 3 to $|V|/2$ where $|V|$ was up to 60 in all three cases. The following characteristics are reported:

- max dev: maximum error from the best known feasible solutions (in %),
- av dev: average deviation from the best known feasible solutions (in %),
- bnd dev: average deviation from the lower bounds calculated by DUASC (in %),
- best: number of best results with respect to the best known feasible solutions (in %).

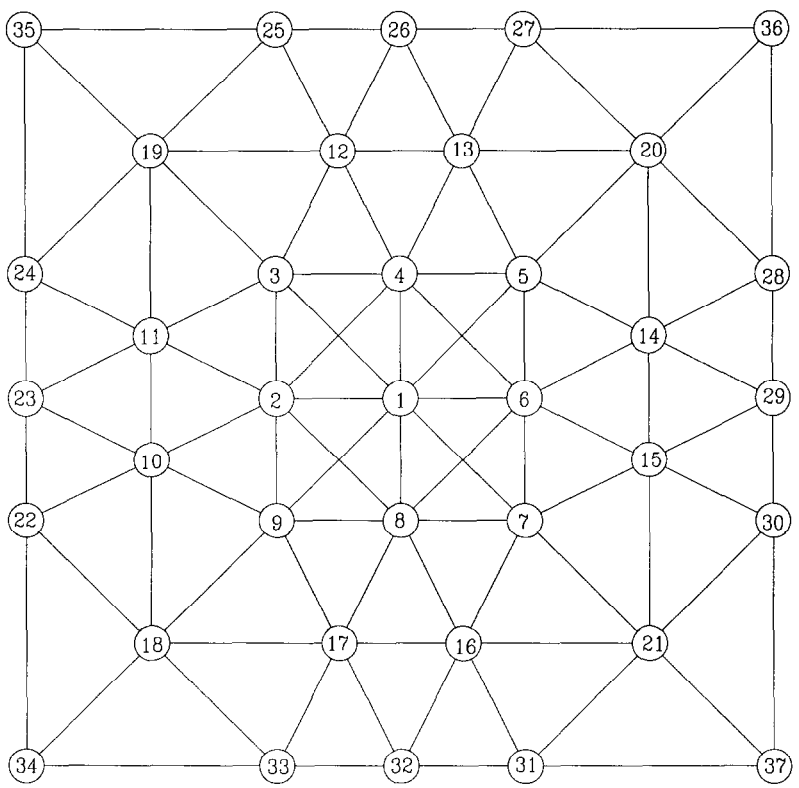

Fig. 1.

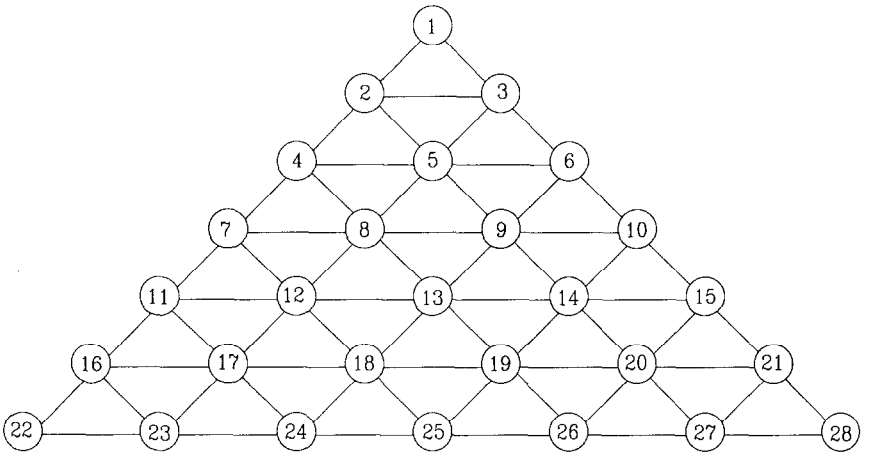

Fig. 2.

All these values will give some information on the quality of obtained solutions. With respect to bnd dev each algorithm may be analyzed by itself. The other values will analyze the average and the worst case performance (av dev, max dev). In addition we are interested in how often each algorithm finds the best known feasible solution.

Table 5 gives evidence that the simplest algorithms SPATH and MST + P are outperformed with respect to all criteria and may not further be taken into consideration. Even ARINS, ARINS-I, and SDISTG come out with remarkably worse

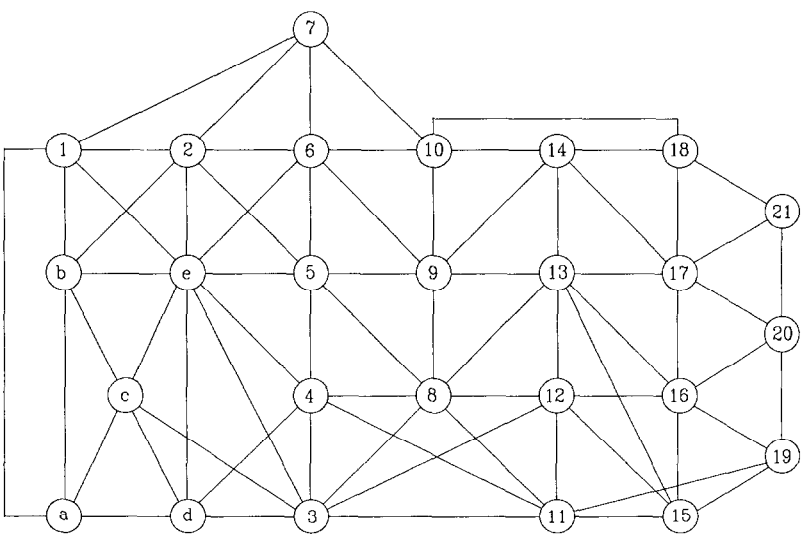

Fig. 3.

Table 5.
Computational results

	Graphs: Figs. 1-3, no. of examples: 906			
Algorithm	max dev	av dev	bnd dev	best
SPATH	96.61	17.24	17.33	13.91
MST+P	141.67	9.92	9.99	18.19
SDISTG	27.27	2.14	2.20	45.47
ARINS	54.02	5.12	5.19	33.76
ARINS-I	54.02	3.00	3.06	46.81
CHINS	15.07	1.00	1.05	63.11
CHINS-I	14.07	0.70	0.75	71.98
CHINS-A	6.98	0.20	0.24	84.85
CHINS-A-I	6.98	0.14	0.15	90.30
CHINS-3	14.07	0.42	0.45	79.99
CHINS-3-I	14.07	0.39	0.42	81.31
HEUM	12.10	0.49	0.52	73.60
HEUM-I	12.10	0.32	0.36	81.32
HEUM-3	14.07	0.57	0.62	73.25
HEUM-3-I	14.07	0.40	0.44	80.89
DUASC	17.27	0.50	0.55	82.87
DUASC-I	10.94	0.26	0.29	88.10

results than all other remaining heuristics. Since ARINS is a simplified version of CHINS, with nearly the same computation times, even these two algorithms need not be considered in the sequel. These arguments are strengthened by all subsequent tests, so for clarity in the presentation of results we may restrict ourselves to the seven remaining algorithms together with the improved versions. (SDISTG remains within consideration and presentation because of its extensive discussion in the literature. A separate improvement version of SDISTG will not be considered because this algorithm includes such a procedure by itself.) However, to have a greater data base (probably on best known feasible solutions) all 17 algorithms have been run through all experiments.

In what follows we are going to describe the generated data before reporting on the experiments.

Random data (real valued). To get results comparable to those of Rayward-Smith and Clare [26] a comprehensive set of test problems has been generated using the scheme outlined above (see e.g. Table 3). The numbers of vertices and edges are chosen according to $|V| \in \{20, 30, 40, 50, 60\}$ and $|E| = i \cdot |V|$ for all $i = 4, 5, \ldots$ with the restriction $|E| \leq 400$ with respect to space considerations (when running all 17 algorithms within the same program). For $|V| = 20$ we have $|E| \leq 190$. Edge weights are real numbers uniformly distributed in the range [1, 10]. Basic vertices are chosen from $|Q| = 5$ up to $|Q| = |V| - 5$ with a stepsize of 5. For each combination of values a sample of 50 experiments has been drawn.

Random data (integer valued). Data in these experiments are chosen in the same way as real-valued random data with the difference that edge weights are integer numbers uniformly distributed in the range [0, 100].

Euclidean graphs. Data in these experiments are graphs with vertices embedded in the plane and edge lengths equal to L_2 distances.

The numbers of vertices are chosen from $|V| = 10$ up to $|V| = 35$ with a stepsize of 5. The real-valued coordinates are uniformly distributed within [0, 100] each. All graphs are complete with edge weights given by Euclidean distances (with all calculations in DOUBLE PRECISION). Basic vertices are chosen according to $|Q| \in \{3, 5, 7, 10, 12, 15, 20, 25, 30\}$ (as far as possible) and a number of 50 experiments has been run on each combination of $|V|$ and $|Q|$.

Grid graphs. In grid graphs within the two-dimensional plane we are given a number $|Q|$ of basic vertices. The real-valued x- and y-coordinates of the basic vertices are uniformly distributed within [0, 100]. The grid graph is induced by the set of basic vertices by running a vertical line and a horizontal line through each basic vertex and retaining the finite segments between interconnections of these lines with rectilinear distances as weights.

Basic vertices are chosen from $|Q| = 3$ up to $|Q| = 9$ with a stepsize of 1. This results in graphs with up to $|V| = 100$ and $|E| = 180$. For each number of basic vertices again a sample of 50 experiments has been drawn.

We are mainly interested in two questions which may give some insight in the performance of the algorithms:

- What is the influence of the (relative) number of basic vertices?
- What is the influence of the density of the graphs?

Following these questions all results have been observed and classified according to two aspects:

- $|Q|/|V|$: relative frequency of basic vertices.
  All Euclidean and random data results have been divided into four intervals with respect to the underlying data:

  $$|Q|/|V| \in]0.0, 0.25], \ldots,]0.75, 1.0].$$

  For grid graphs, if all basic vertices lie on different horizontal and vertical lines, we have the relationship $|Q|/|V| = 1/|Q|$, i.e., all values are within $]0.0, 0.\bar{3}]$.
- dens $:= 2 \cdot |E|/|V| \cdot (|V| - 1)$: density of the underlying graph.
  All random data results have been divided into five intervals:

  $$\text{dens} \in]0.0, 0.2], \ldots,]0.8, 1.0].$$

  The generation of Euclidean graphs leads to dens $= 1.0$ in all cases whereas for grid graphs we have dens $= 4/(|Q|^2 + |Q|)$, i.e., all values are within $]0.0, 0.\bar{3}]$.

Remark. In all cases again the simplest algorithms SPATH, MST + P, and ARINS behave worst. The average deviation with respect to ARINS is best ($\approx 5\%$) for low ratios $|Q|/|V|$ whereas MST + P gets the best results when $|Q|/|V|$ approaches 1.

In random graphs it may be very successful to use an improvement procedure with each of the six best heuristics, so in the following we compare CHINS-I, CHINS-A-

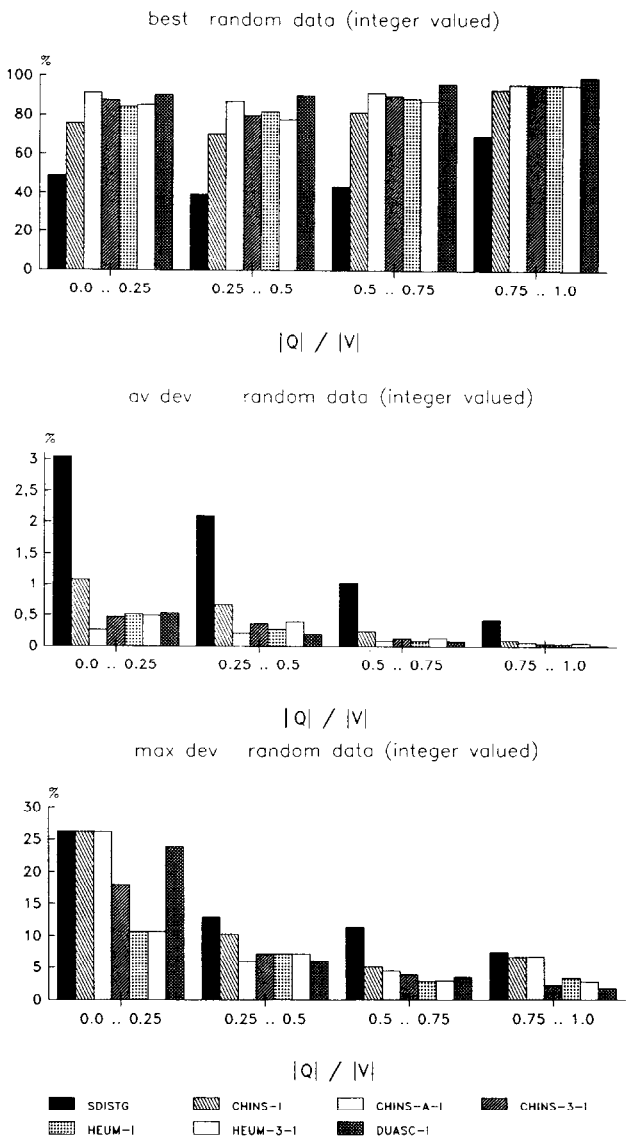

Fig. 4. Results on integer-valued random graphs ($|Q|/|V|$).

I, CHINS-3-I, HEUM-I, HEUM-3-I, and DUASC-I together with SDISTG. In about 90% of all tests with random data this may lead to improvements, which, however, are mostly not very high.

In the sequel the results will be illustrated by figures with respect to best, av dev, and max dev according to the above mentioned criteria. Whenever there is not too much overlap we will draw some lines, and give corresponding diagrams otherwise.

Random data (integer valued) (see Figs. 4, 5). SDISTG is clearly outperformed by all other (six) heuristics. All algorithms get better as the relative number of basic vertices increases, which is emphasized by decreasing av dev values in these cases (compare also max dev).

The bound deviation bnd dev of the best known feasible solutions in these experiments is given as 0.03% so all algorithms behave quite good. Best results even show decreasing performance for decreasing $|Q|/|V|$. However, for $|Q|/|V| \in]0.0, 0.25]$ best results slightly increase because of more simplicity of the problems for small $|Q|$.

To sum up the results shown in Fig. 4, DUASC-I comes up best for $|Q|/$

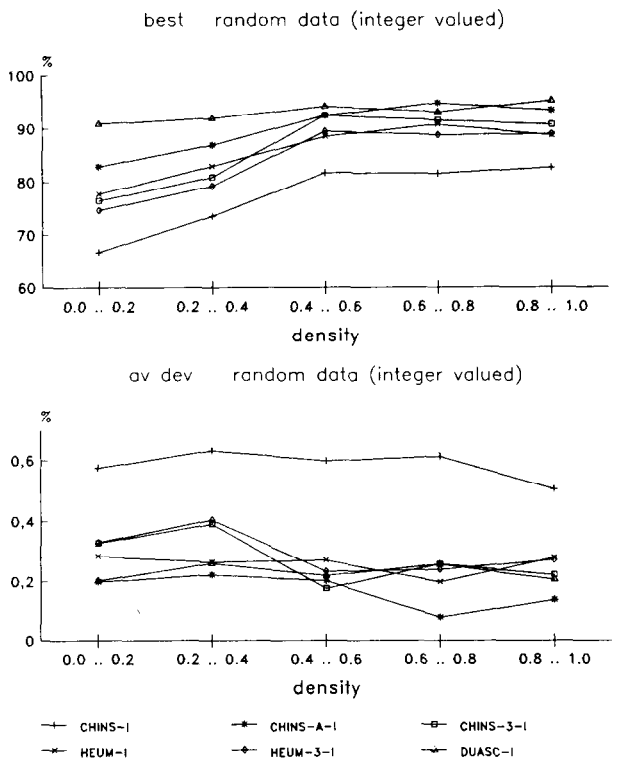

Fig. 5. Results on integer-valued random graphs (density).

$|V|>0.25$. It produces most best results with lowest av dev (and comparable max dev). For small $|Q|/|V|$ the more sophisticated insertion and the heuristic measure methods may be preferred. All results also indicate that the modifications CHINS-A(-I) as well as CHINS-3(-I) of CHINS are worth being considered removing the shortcoming of insertion methods against heuristic measure as reported above (cf. [26]).

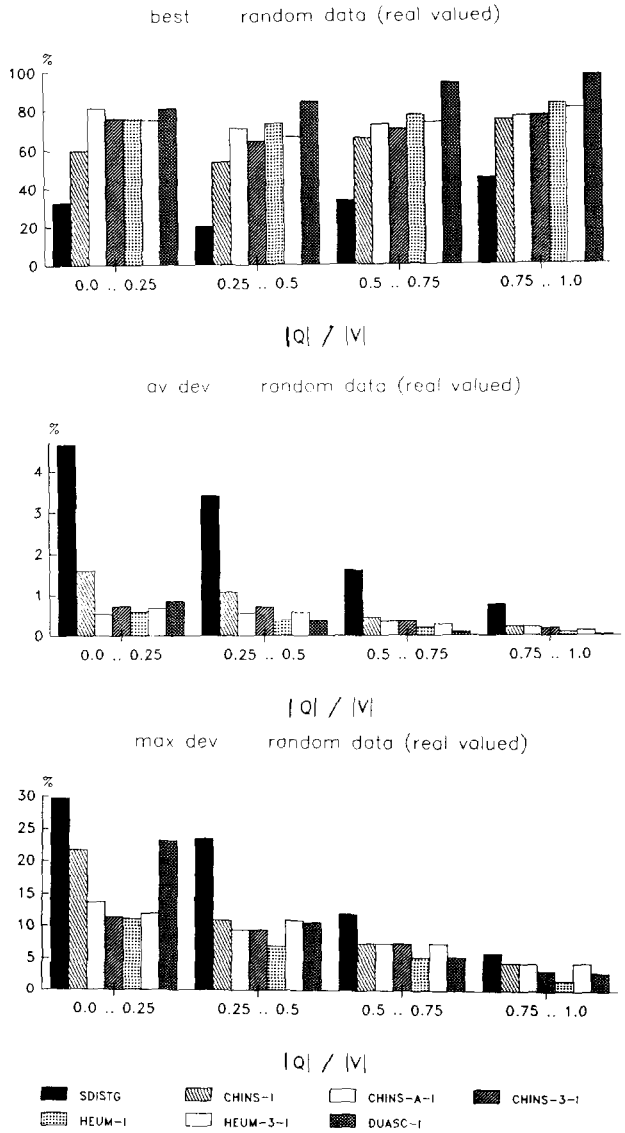

Fig. 6. Results on real-valued random graphs ($|Q|/|V|$).

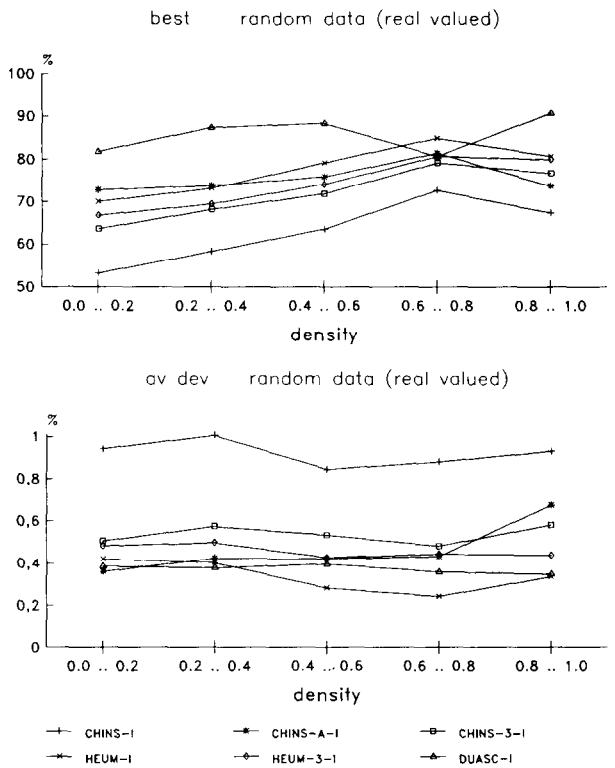

Fig. 7. Results on real-valued random graphs (density).

A little difference may be reported with respect to density. In all intervals DUASC-I stays about 93% best results whereas all other algorithms show increasing rates with the density increasing up to 0.8 (for clarity SDISTG has been omitted in Fig. 5). For dens $\in]0.4, 0.8]$ heuristic measure and CHINS-3-I are comparable to DUASC-I but CHINS-A-I should be preferred for dens ≥ 0.6.

Density results do not care on the relative frequency of basic vertices since for different intervals of $|Q|/|V|$ the principle proportion of the algorithms among one another with respect to dens behaves the same as shown in Fig. 5. As max dev in connection with dens does not give any further information such diagrams have been omitted. The question arises whether the performance of heuristics is influenced by the degree of variation of edge weights. Our results with weights from [0, 100] give the tendency which is strengthened for weights from [0, 10] and approaches results of real-valued data for [0, 1000] which will be reported next.

Random data (real valued) (see Figs. 6, 7). For $|Q|/|V|$ the same results may be reported as for integer values with the difference that all results get worse, and the superiority of DUASC-I with respect to best results becomes more obvious. Bnd dev for the best known feasible solution is 0.1% (on an average over all 50 experiments).

Density results in Fig. 7 again give for DUASC-I the highest proportion with respect to best. However, for sparse graphs the corresponding values are below 85%. For graphs with dens>0.4, HEUM-I outperforms all other heuristics with respect to av dev whereas DUASC-I behaves second best.

If graphs get denser most algorithms nearly behave the same with respect to av dev but improve with respect to the number of best solutions. We did not recognize any influence of the variation of edge weights which is due to dealing with real values.

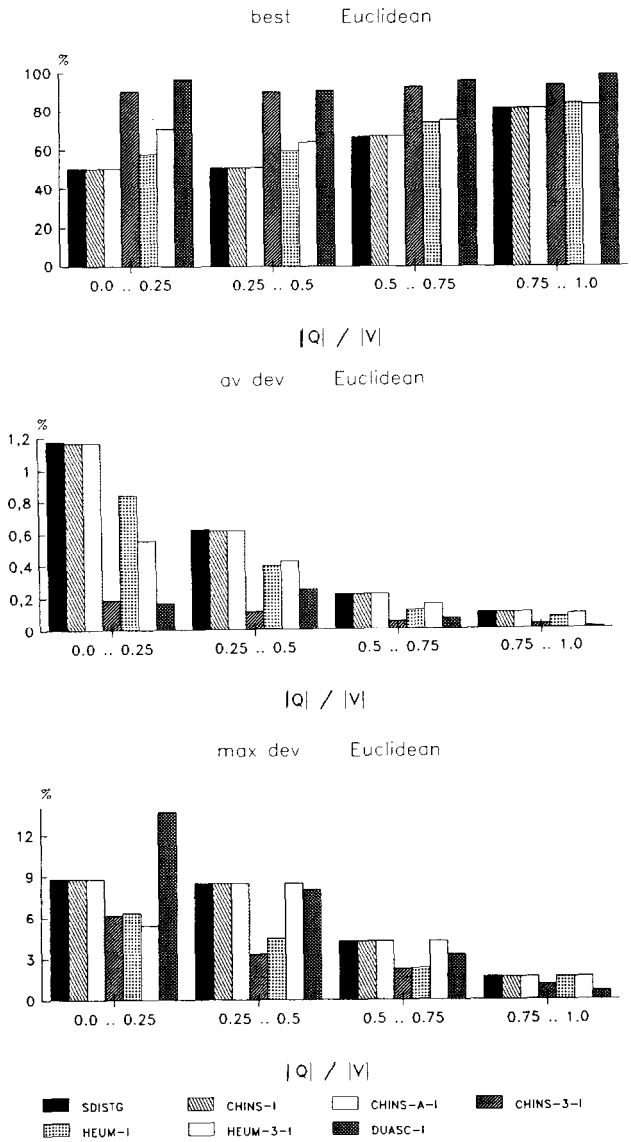

Fig. 8. Results on Euclidean graphs ($|Q|/|V|$).

Euclidean graphs (Fig. 8). For $|Q| = 3$, CHINS-3(-I), HEUM-3(-I), and DUASC(-I) found optimal solutions in all cases and outperform the other algorithms (these examples are omitted in Fig. 8).

SDISTG, CHINS(-I), and CHINS-A(-I) in general should come out with the same solutions worse than the remaining algorithms. This is due to the fact that Steiner vertices may only come into consideration if at least one such vertex lies on the direct connection between two basic vertices. Similar arguments also show that I-MST + P mostly does not lead to any improvements.

In all cases DUASC(-I) and CHINS-3(-I) behave best with respect to best and av dev though DUASC(-I) may have quite bad results (see max dev) for $|Q|/|V| \leq 0.5$. With respect to av dev CHINS-3(-I) gives slightly better results than DUASC(-I) in $]0.25, 0.75]$. The opposite holds for the complementary intervals. Next best HEUM(-I) and HEUM-3(-I) give nearly the same values with advantages for the first if $|Q|/|V| \leq 0.5$ and for the latter if $|Q|/|V| > 0.5$. The bound deviation of the best known feasible solutions in these experiments is given as 0.02%.

In general results get better for all algorithms when $|Q|/|V|$ approaches 1. Nearly the same observations hold for additional tests with rectilinear instead of Euclidean distances.

Grid graphs (Fig. 9). The results for grid graphs greatly differ from the preceding results. All heuristics come up with fairly good results for $|Q| = 3$ and $|Q| = 4$ (i.e., graphs with $|Q|/|V|$ equal to $0.\bar{3}$ and 0.25 and dens equal to $0.\bar{3}$ and 0.2, respectively) with CHINS-A(-I) getting optimal solutions in these cases. For increasing numbers of basic vertices results get worse for all algorithms, especially for DUASC-I. The level of best results falls below 60% and av dev turns out to be much higher than for the other classes of graphs. The bound deviation of the best known feasible solutions in these experiments is given as 0.6%.

The remaining results nearly give the same behaviour as discussed for the other classes but on a lower level and with DUASC-I as an exception. CHINS-A-I may be recommended as the best algorithm for all values of $|Q|$, for $|Q| \geq 7$ together with CHINS-3-I.

DUASC-I gives second-worst results, behind SDISTG. Careful analysis of results leading to max dev values shows a shortcoming with respect to grid graphs which may be generalized to graphs where a lot of edge weights are exactly the same. In this case it may happen that step (2) of DUASC produces a solution which mostly includes an optimal or near optimal solution. In our implementation of step (3), (3.1)–(3.3), no rules for breaking ties have been included. This may lead to bad results. Preliminary tests show that suitable reduction techniques included in step (3) may help to step out of this shortcoming.

To sum up our results several observations may be drawn. In general algorithms using cheapest insertion, heuristic measure, or dual ascent together with a simple improvement procedure are outperforming the four more or less "simple" algorithms

SPATH, MST + P, ARINS, and SDISTG. With respect to the lower bound DUASC is the most interesting algorithm. It is worth mentioning that in most cases where DUASC was outperformed by some of the other algorithms the resulting graph after step (2) included the optimal solution, and quite simple reduction techniques will result in that solution.

As a recommendation several of these simple algorithms should be applied simultaneously to take the best of the achieved results. In addition some simple modifications on existing algorithms may give improvements with respect to solu-

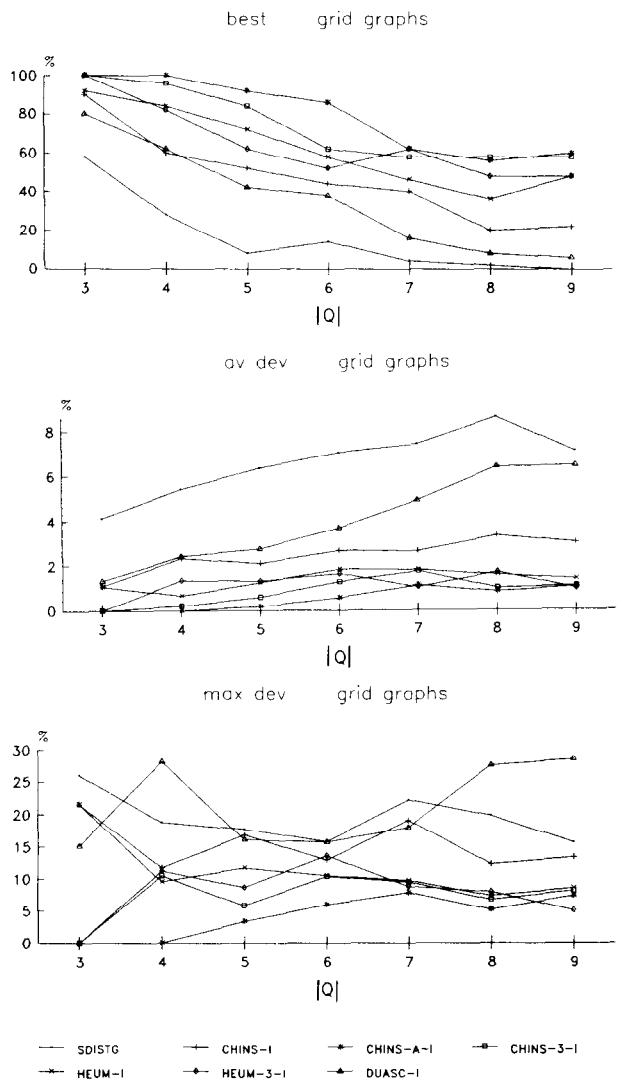

Fig. 9. Results on grid graphs ($|Q|/|V|$).

tion quality (e.g. CHINS-A-I, instead of CHINS or CHINS-I). All but four algorithms (CHINS-3, CHINS-3-I, DUASC, DUASC-I) have running times which even on a PC (AT, 10 Mhz) lie within 0 and 2 seconds for problems with up to 60 vertices and 400 edges. For most examples running times of CHINS-3, CHINS-3-I, DUASC, and DUASC-I are even comparable. In the worst case times of CHINS-3-I and DUASC-I are below 90 seconds, too.

Our results concerning the modifications of cheapest insertion are strengthened in a very recent working paper [39] where SDISTG, HEUM, and CHINS are compared with some repetitive applications of CHINS while using some simple preprocessing as proposed above. ([39] got known to the author in the final stage of the refereeing process leaving an inclusion of these algorithms to the computational comparison for further research.)

6. Conclusions

In this paper we have reviewed the literature on heuristic algorithms for Steiner's problem in graphs. Computational experiments indicate that various algorithms may be recommended together with a simple improvement procedure. These algorithms involve ideas from cheapest insertion, heuristic measure, and dual ascent. Most of the algorithms are quite easy to implement and may run on a PC with tolerable computation times.

What remains open are developments of efficient modifications of some of the algorithms as well as some more theoretical investigations especially with the dual ascent algorithm (one such aspect is treated in [13]). Again the behaviour of the set covering approach is to be explored. Some computational experiences with e.g. an interchange heuristic may also be worth exploring. In addition the impact of efficient preprocessing routines in combination with different heuristics should be investigated.

However, the most important aspect lies in testing the algorithms described in this paper on (large scale) real world problems e.g. from VLSI design.

References

[1] Y.P. Aneja, An integer linear programming approach to the Steiner problem in graphs, Networks 10 (1980) 167–178.

[2] A. Balakrishnan and N.R. Patel, Problem reduction methods and a tree generation algorithm for the Steiner network problem, Networks 17 (1987) 65–85.

[3] J.E. Beasley, An algorithm for the Steiner problem in graphs, Networks 14 (1984) 147–159.

[4] J.E. Beasley, An SST-based algorithm for the Steiner problem in graphs, Networks 19 (1989) 1–16.

[5] N.P. Chen, New algorithms for Steiner tree on graphs, in: Proceedings of the IEEE International Symposium on Circuits and Systems (1983) 1217–1219.

[6] E.A. Choukhmane, Une heuristique pour le probleme de l'arbre de Steiner, RAIRO Rech. Opér. 12 (1978) 207–212.

[7] C.W. Duin and A. Volgenant, An edge elimination test for the Steiner problem in graphs, Oper. Res. Lett. 8 (1989) 79–83.

[8] C.W. Duin and A. Volgenant, Reduction tests for the Steiner problem in graphs, Networks 19 (1989) 549–567.

[9] L.R. Foulds and V.J. Rayward-Smith, Steiner problems in graphs: algorithms and applications, Engrg. Optimization 7 (1983) 7–16.

[10] L. Guyard-Daher, Le problème de l'arbre de Steiner, Ph.D. thesis, L'École Nationale Supériore des Télécommunications, Paris (1985).

[11] J. Hesser, R. Männer and O. Stucky, Optimization of Steiner trees using genetical algorithms, in: J.D. Schaffer, ed., Proceedings of the Third International Conference on Genetic Algorithms (Morgan Kaufmann, San Mateo, 1989) 231–236.

[12] A. Iwainsky, Some notes on the Steiner tree problem in graphs, in: A. Iwainsky, ed., Optimization of Connection Structures in Graphs (Central Institute of Cybernetics and Information Processes, Berlin, 1985) 57–73.

[13] A. Jain, The Steiner problem on directed graphs: a probabilistic analysis of the duality gap, Paper presented at the NATO Advanced Research Workshop on Topological Network Design, Copenhagen (1989).

[14] B. Korte and R.H. Möhring, Zur Bedeutung der diskreten Mathematik für die Konstruktion hochintegrierter Schaltkreise, in: R. Henn, ed., Technologie, Wachstum und Beschäftigung – Festschrift für Lothar Späth (Springer, Berlin, 1987) 582–596.

[15] L.T. Kou, On efficient implementation of an approximation algorithm for the Steiner tree problem, Acta Inform. 27 (1990) 369–380.

[16] L.T. Kou and K. Makki, An even faster approximation algorithm for the Steiner tree problem in graphs, Congr. Numer. 59 (1987) 147–154.

[17] L. Kou, G. Markowsky and L. Berman, A fast algorithm for Steiner trees, Acta Inform. 15 (1981) 141–145.

[18] J.B. Kruskal, On the shortest spanning subtree of a graph and the travelling salesman problem, Proc. Amer. Math. Soc. 7 (1956) 48–50.

[19] L. Kucera, A. Marchetti-Spaccamela, M. Protasi and M. Talamo, Near optimal algorithms for finding minimum Steiner trees on random graphs, in: J. Gruska, B. Rovan and J. Wiedermann, eds., Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 233 (Springer, Berlin, 1986) 501–511.

[20] H.T. Lau, Combinatorial Heuristic Algorithms with FORTRAN, Lecture Notes in Economics and Mathematical Systems 280 (Springer, Berlin, 1986).

[21] K. Mehlhorn, A faster approximation algorithm for the Steiner problem in graphs, Inform. Process. Lett. 27 (1988) 125–128.

[22] O. Palma-Pacheco, Contribution to solve the Steiner problem in a directed graph: a heuristic method, Ph. D. thesis, COPPE, Rio de Janeiro (1985) (in Portuguese).

[23] J. Plesnik, A bound for the Steiner tree problem in graphs, Math. Slovaca 31 (1981) 155–163.

[24] R.C. Prim, Shortest connection networks and some generalizations, Bell System Tech. J. 36 (1957) 1389–1401.

[25] V.J. Rayward-Smith, The computation of nearly minimal Steiner trees in graphs, Internat. J. Math. Ed. Sci. Tech. 14 (1983) 15–23.

[26] V.J. Rayward-Smith and A. Clare, On finding Steiner vertices, Networks 16 (1986) 283–294.

[27] C. Schiemangk, Thermodynamically motivated simulation for solving the Steiner tree problem and the optimization of interacting path systems, in: A. Iwainsky, ed., Optimization of Connection Structures in Graphs (Central Institute of Cybernetics and Information Processes, Berlin, 1985) 91–120.

[28] M. Servit, Heuristic algorithms for rectilinear Steiner trees, Digital Processes 7 (1981) 21–32.

[29] M. Shaohan, The Steiner tree problem on graph and its heuristic algorithm, Chinese J. Comput. 8 (1985) 237–239 (in Chinese).

[30] G.F. Sullivan, Approximation algorithms for Steiner tree problems, Tech. Rep. 249, Department of Computer Science, Yale University, New Haven, CT (1982).
[31] H. Takahashi and A. Matsuyama, An approximate solution for the Steiner problem in graphs, Math. Japon. 24 (1980) 573–577.
[32] S. Voß, A reduction based algorithm for the Steiner problem in graphs, Methods of Operations Research 58 (Athenäum/Hain/Hanstein, Königstein, 1989) 239–252.
[33] S. Voß, Steiner-Probleme in Graphen (Hain, Frankfurt/Main, 1990).
[34] S.M. Wang, A multiple source algorithm for suboptimum Steiner trees in graphs, in: H. Noltemeier, ed., Proceedings of the International Workshop on Graphtheoretic Concepts in Computer Science (Trauner, Würzburg, 1985) 387–396.
[35] B.M. Waxman and M. Imase, Worst-case performance of Rayward-Smith's Steiner tree heuristic, Inform. Process. Lett. 29 (1988) 283–287.
[36] P. Widmayer, Fast approximation algorithms for Steiner's problem in graphs, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe (1986).
[37] P. Widmayer, On approximation algorithms for Steiner's problem in graphs, in: G. Tinhofer and G. Schmidt, eds., Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 246 (Springer, Berlin, 1987) 17–28.
[38] P. Winter, Steiner problem in networks: a survey, Networks 17 (1987) 129–167.
[39] P. Winter and J. MacGregor Smith, Path-distance heuristics for the Steiner problem in undirected networks, Algorithmica, to appear.
[40] R.T. Wong, A dual ascent approach for Steiner tree problems on a directed graph, Math. Programming 28 (1984) 271–287.
[41] Y.F. Wu, P. Widmayer and C.K. Wong, A faster approximation algorithm for the Steiner problem in graphs, Acta Inform. 23 (1986) 223–229.